Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matija Vižintin

Varnost v spletni aplikaciji Dnevniki odjema odjemalca

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Mojca Ciglarič

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matija Vižintin, z vpisno številko **63050128**, sem avtor diplomskega dela z naslovom:

Varnost v spletni aplikaciji Dnevniki odjema odjemalca

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 24. aprila 2014 Podpis avtorja:

Zahvalujem se doc. dr. Mojci Ciglarič za strokovno pomoč pri izdelavi diplomske naloge, sodelavcem za pomoč in nasvete pri izdelavi spletne aplikacije, Sandri za lektoriranje in staršem za podporo v času študija.

Kazalo

Slike

Tabele

Izvorna koda

Povzetek

Večje slovensko podjetje, ki se ukvarja z energijo, je potrebovalo aplikacijo, s katero bi svojim odjemalcem omogočilo vnos podatkov o odjemih zemeljskega plina na posameznih merilnih mestih. Zanj smo razvili spletno aplikacijo za izvedbo avtomatiziranega pridobivanja podatkov. Avtomatizirana rešitev pridobivanja podatkov o odjemih je morala biti izdelana z namenom, da bi odjemalci podatke o odjemih posredovali družbi na elektronski in varen način. Velik poudarek je na varnosti aplikacije, ki mora biti zagotovljena na vseh nivojih, saj vsebuje zaupne podatke in je izpostavljena na medmrežju. Opisali bomo tehnologije, ki so bile uporabljene pri realizaciji rešitve, navedli bomo, kateri varnostni protokoli so implementirani, in pojasnili, na kakšen način zagotavljajo varnost. Cilj projekta je varna spletna aplikacija, ki je integrirana z obstoječim sistemom, z njim varno komunicira preko spletne storitve ter omogoča dostop in vnos odjemov samo avtoriziranim uporabnikom.

Ključne besede: varnost, Java EE, Seam, JSF, spletne storitve, SSL

Abstract

A major Slovenian public utility needed an application that would facilitate its customers to insert gas consumption data on its measurement places. We developed a web application to automatically retrieve gas consumption data. This automated data retrieval solution was developed to allow consumers to provide consumption data to the system operator in an electronic and secure way. We put a big effort in application security aspect, since it had to be assured on every communication layer, because confidential information should not be exposed on internet. We will describe technologies that were used realizing this application, which security protocols were implemented and how to provide security. This project aims for a secure web application that is integrated with the existing system, it securely communicates via web service and allows access and consumption data insertion to authorized users only.

Ključne besede: security, Java EE, Seam, JSF, web services, SSL

Seznam kratic

API (ang. Application Programming Interface) - aplikacijski programski vmesnik

DMZ (ang. Demilitarized Zone) - nezaščiten del omrežja, ki je navadno izpostavljeno na medmrežje

EJB (ang. Enterprise JavaBeans) - strežniška arhitektura za modularno izdelavo poslovnih aplikacij

HTTP (ang. Hypertext Transfer Protocol) - protokol za prenos hiperbesedil na spletu

HTTPS (ang. Hypertext Transfer Protocol Secure) - HTTP protokol, ki uporablja SSL ali TLS protokol za varno komunikacijo med strežnikom in odjemalcem

Java EE (ang. Java Platform, Enterprise Edition) - poslovna različica javanske platforme

Java ME (ang. Java Platform, Micro Edition) - različica javanske platforme za vgrajene sisteme

Java SE (ang. Java Platform, Standard Edition) - standradna raličica javanske platforme

JDK (ang. Java Development Kit) - skupek orodij za razvoj, razhroščevanje in nadziranje javanskih aplikacij

JRE (ang. Java Runtime Environment) - izvajalno okolje za javanske aplikacije

JSF (ang. JavaServer Faces) - tehnologija za izgradnjo uporabniških vmesnikov spletnih strani, ki temeljijo na komponentah

JVM (ang. Java Virtual Machine) - javanski virtualni stroj

ORM (ang. Object-relational mapping) - objektno relacijska preslikava

POJO (ang. Plain Old Java Object) - navaden javanski objekt

SOAP (ang. Simple Object Access Protocol) - specifikacija protokola za izmenjavo strukturiranih podatkov

SQL (ang. Structured Query Language) - strukturirani povpraševalni jezik za delo s podatkovnimi bazami

SSL (ang. Secure Sockets Layer) - kriptografski protokol, ki omogoča varno komunikacijo na medmrežju

TLS (ang. Transport Layer Security) - kriptografski protokol, naslednik SSL, ki omogoča varno komunikacijo na medrežju

URL (ang. Uniform Resource Locator) - naslov spletnih strani v svetovnem spletu

WSDL (ang. Web Services Description Language) - opisni jezik, ki bazira na XML standardu in se uporablja za opisovanje funkcionalnosti spletnih storitev

Poglavje 1

Uvod

1.1 Motivacija

Sistemski operater prenosnega omrežja je vzpostavil proces avtomatskega zbiranja podatkov o odjemu iz naprav za merjenje pretoka plina ter njihove validacije, proces prenosa v obračun in proces izdelave obračuna prenosa. Proces je podprt z informacijskimi sistemi SCADA, TELEREADING, VALIDACIJA in OPZP.

Kljub avtomatskemu zbiranju podatkov, pa je nekaj merilnih mest izvzetih. Večinoma gre pri tem za virtualna merilna mesta (merilna mesta, ki pomenijo odjem bilančnih podskupin na mestnih vratih). Za ta merilna mesta vpisujejo uporabniki podatke odjema v posebne obrazce, katere jim posreduje sistemski operater. Izpolnjene obrazce prepišejo v službi za komercialo in regulacijo sistemskega operaterja. Tak postopek povzroča časovne zamike pri pridobivanju podatkov, dodatno delo in povečano možnost napak.

Ideja za izboljšavo je tako izdelava aplikacije, ki nadomesti dosedanji ročni vnos podatkov. Aplikacija mora biti integrirana z dosedanjimi sistemi in v njih nevidno nadomestiti do sedaj ročno opravljeno delo. Ker bo nov sistem na voljo vsem odjemalcem, mora biti dostopen na medmrežju. Z izpostavitvijo aplikacije na medmrežje bo potrebno poskrbeti za varnost in zaščititi vse občutljive informacije v dosedanjih sistemih, ki delujejo v internem zapr-

tem okolju. Tako mora aplikacija poskrbeti, da neavtoriziran uporabnik ne pride do internih informacij, avtoriziran uporabnik pa lahko dostopa samo do podatkov, do katerih je upravičen. Vse te zahteve pomenijo, da bo v aplikacijo potrebno vgraditi več nivojsko varnostno arhitekturo, ki bo preverjala pristnost uporabnika, njegove pravice za dostop do aplikacije, pravice za dostop do podatkov in vnos odjemov ter zagotovila varno komunikacijo med dosedanjimi sistemi in novim sistemom za avtomatski vnos dnevnikov odjema odjemalca.

Pridobitev takšnega sistema je, da z ustreznimi nivoji pravic zagotovimo dostop do svojih podatkov samo avtoriziranim odjemalcem, zagotovimo točnost prihoda podatkov, za katere so sedaj odgovorni odjemalci sami, preprečimo kakršno koli možnost napake na strani sistemskega operaterja prenosnega omrežja pri ročnem vnašanju vrednosti v sistem in hkrati lahko natančno beležimo ves pretok informacij med odjemalcem in sistemskim operaterjem.

1.2 Kaj je avtomatizacija procesa in kaj prinaša

Avtomatizacija je uporaba sistemov in informacijske tehnologije na način, da zmanjšamo potrebo po človeških virih. Sedaj to delo opravijo avtomatizirani sistemi, ki so bili razviti s tem namenom. S tem človeške vire sprostimo iz monotonega dela in jih lahko usmerimo na razvoj, namestitev in vzdrževanje avtomatiziranih procesov. Poleg tega zmanjšamo možnost napak, ki se pojavi pri monotonem delu, ki ga delavci opravljajo rutinsko, kar povzroči padec koncentracije. Dobro narejeni in testirani sistemi opravijo te naloge vedno v bistveno krajšem času in brez napak.

Poleg tega ima avtomatizacija procesa tudi svoje slabe strani. Za uvedbo takšnega procesa na začetku potrebujemo kar precejšen finančni vložek. Poleg tega so tudi stroški raziskav in razvoja lahko nepredvidljivi. Na začetku ne vemo natančno, koliko bodo stroški znašali in koliko dodatnih težav in zahtev

se lahko pojavi med razvojem samim. Avtomatiziran sistem ima omejen nabor sposobnosti in znanj, kar pomeni, da je lahko izpostavljen varnostnemu tveganju.[1]

Prav zato je potrebno v takšnih sistemih poskrbeti za varnost, še posebej pri tistih, ki so odprti v medmrežje in s tem potencialnim zlonamernim napadom.

1.3 Zgradba diplomskega dela

Namen diplomske naloge je predstaviti varnostne vidike aplikacije, ki je izpostavljena na medmrežju. Na primeru spletne aplikacije Dnevniki odjema odjemalca želimo prikazati nevarnosti, ki pretijo na tak sistem, ki je dostopen širši množici, in predstaviti rešitve za preprečevanje takšnih groženj.

Ko govorimo na temo varnosti v spletnih aplikacijah, najprej pomislimo na vdore v spletne strani, krajo podatkov o kreditnih karticah in napade denial-of-service. Druge nevarnosti, ki to spremljajo, so še virusi in trojanski konji, ki prav tako nepooblaščeno poskušajo pridobiti zaupne informacije. Druge nevarnosti, ki jih ponavadi spregledamo, so še zlonamerni administratorji, nezadovoljni zaposleni in naključni uporabniki, ki po nesreči pridejo do zaupnih podatkov. Največji problem pa je ravno nevednost in nezadostno znanje. Rešitev je v stalnem izboljševanju celotnega procesa, zavedanju nevarnosti, izobraževanju ljudi in nadgrajevanju varnostnih mehanizmov.[2]

Na začetku predstavimo projekt in kašne so bile naročnikove zahteve. Tehnologije, ki so bile uporabljene za nastanek aplikacije, so v večji meri pogojene z integracijo z obstoječo aplikacijo in v okviru tega bomo najprej predstavili vse potencialne varnostne luknje, ki jih predpisana tehnološka izvedba integracije in posledično zahtevani način komunikacije med aplikacijama predstavlja. Natančno bodo opisani pretok podatkov, želena interakcija uporabnika s sistemom in akcije, ki se posledično izvajajo v ozadju. Poleg tega bomo še opisali način avtentikacije in avtorizacije, s katerima dobi uporabnik vpogled do vsebin, za katere ima pravice. Predstavljen bo tudi meha-

nizem, ki ob izpadu aplikacije, katera streže podatke, uporabniku še vedno omogoča normalno delo v mejah, ki preprečujejo zlorabe.

V 3. poglavju predstavimo tehnologije, ki so bile uporabljene za izgradnjo spletne aplikacije oz. na katerih aplikacija deluje. Pri vseh tehnologijah je poudarek na varnostnem vidiku oz. kako te tehnologije preprečujejo potencialne varnostne grožnje. Tako aplikacijski strežnik omogoča prijavo v sistem in zagotavlja preverjanje avtorizacije za dostop do poslovne logike. Skupaj z dodatno razvitim modulom zagotovimo zgoščanje gesel. Spletne storitve prav tako zahtevajo avtentikacijo in dodatno preverjajo privilegije sistema, s katerimi preprečimo dostop drugim zlonamernim sistemom. Ogrodje Seam pomaga pri zagotavljanju varnosti pri prijavi v spletno aplikacijo, preverja navigacijo po spletni aplikaciji in preverja vnosna polja.

V 4. poglavju opišemo rešitev problema in izdelavo spletne aplikacije. Poleg tega so opisane varnostne grožnje, ki nastanejo pri takem načinu integracije, in praktične rešitve le-teh. Opišemo, v kateri smeri se prenašajo podatki, kje se šifrira povezava, na katerem nivoju se to zagotavlja in kakšni so mehanizmi, ki to zagotavljajo. Kot zadnje je v tem poglavju opisana konfiguracija sistema, ki je vedno pomembna. Tehnologije nam ponujajo veliko varnostnih rešitev, ki pa ne učinkujejo, če jih ne znano pravilno uporabiti in usposobiti.

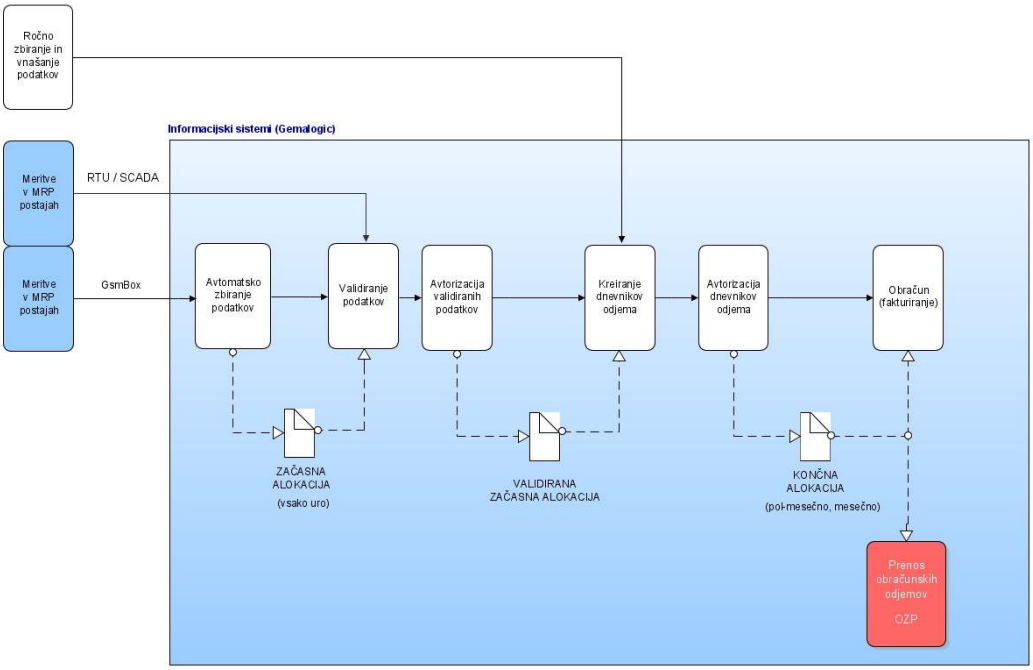
Poglavje 2

Opis problema in zahtev naročnika

2.1 Opis problema

Cilj projekta Dnevniki odjema odjemalca (DOO) je priprava aplikacije in drugih potrebnih komponent, ki bodo omogočale vnos dnevnikov odjema zunanjemu odjemalcu ter spremembe na obstoječi aplikaciji Obračun prenosa zemeljskega plina (OPZP). Slednje so potrebne, da se podatki iz aplikacije DOO prenesejo v aplikacijo OPZP. Odjemalcem bodo na voljo podatki o vnesenih in oddanih dnevnikih odjema.

Opisali bomo obstoječe stanje v podjetju in obstoječe sisteme s podatki, ki so na voljo. Definirali bomo vsebinske, tehnične in varnostne zahteve. Dodatno bomo definirali predpisano arhitekturo sistema, podatkovni tok in način določanja pravic dostopa do dnevnikov odjema za zunanje odjemalce zemeljskega plina.



Slika 2.1: Sistem GemaLogic

2.2 Vsebinske zahteve

2.2.1 Vnos dnevnikov

Uporabniki (odjemalci) vnašajo podatke s pomočjo internetnega brskalnika. Dostop do orodja za vnos dnevnikov se ureja z orodjem Gema Administrator, ki je del informacijskega sistema GemaLogic in se uporablja za administracijo sistema, v aplikaciji Dnevniki odjema odjemalca na strežniku v demilitarizirani zoni (DMZ). V računalništvu se izraz demilitarizirana zona ali krajše DMZ uporablja za del omrežja, ki ni zaščiteno in je navadno izpostavljeno na medmrežju.

Uporabniki imajo možnost vnašati dnevnike odjema za tista merilna mesta, ki jim jih ponudi aplikacija Obračun prenosa zemeljskega plina (s po-

močjo spletne storitve, glej tudi poglavje 2.2.3 Omejitve dostopa na merilnih mestih), in dnevnike, ki še niso bili vneseni v aplikacijo OPZP po drugi poti.

Dnevniki odjema, ki jih vnašajo uporabniki v aplikaciji DOO, obsegajo podatke dnevnih odjemov za obdobje enega meseca.

Za običajna merilna mesta se lahko oziroma morajo (označeni z *) vpisovati naslednji podatki:

- tlak,
- temperatura,
- * številčno stanje plinomera,
- * številčno stanje korektorja,
- * številčno stanje spominske enote,
- * dnevni odjem spominske enote,
- pavšalni odjem v obdobju.

Za virtualna merilna mesta se vpisujeta:

- pavšalni odjem v obdobju,
- * dnevni odjem spominske enote.

Način preračuna odjemov se prenese iz podatkov merilnega mesta iz aplikacije OPZP. Tam se način preračuna na merilnem mestu ureja z orodjem Gema Administrator. Po zaključenem vnosu uporabnik označi dnevnik za pošiljanje. Dokler ni dnevnik dejansko prenesen in ne dobi takega statusa, ga lahko uporabnik še popravlja.

2.2.2 Prenos dnevnikov

Dnevnike, ki so označeni za pošiljanje, posebno opravilo na strežniku OPZP po urniku prenese v aplikacijo OPZP in jih označi kot “prenesene” v aplikaciji DOO in kot “potrjene” v aplikaciji OPZP. Opravilo dostopa do dnevnikov

preko spletne storitve na strežniku DMZ. V primeru napake pri prenosu dnevnika, se ti dnevniki označijo kot "zavrnjeni". Te dnevnike lahko uporabniki ponovno popravijo in pošljejo v aplikacijo OPZP.

2.2.3 Omejitve dostopa na merilnih mestih

Povezava med pravno osebo uporabnika (vnašalca) in merilnimi mesti, za katere lahko uporabnik vnaša dnevnike, so podatki na pogodbah o dostopu, na katerih nastopajo merilna mesta. Pogodba o dostopu je dokument, ki definira lastništvo merilnih mest za dogovorjeno časovno obdobje in mora biti veljaven v obdobju dnevnika.

Pravna oseba uporabnika mora biti lastnik podatkov, plačnik, sklenitelj ali nosilec prenosne kapacitete (PK), ki je definirana z enoto standardni kubični meter na uro (Sm^3/h), na pogodbi o dostopu ali dobavitelj na virtualno merilno mesto, katero v realnosti ne obstaja in se ustvari na podlagi podzakupa druge pravne osebe na resničnem merilnem mestu, in sicer iz bilančnih pogodb, ki definirajo pripadnost merilnega mesta določeni bilančni skupini za dogovorjeno časovno obdobje, v obdobju dnevnika. Vloga pravne osebe na pogodbi o dostopu mora biti enaka vlogi, ki je izbrana za vnašalca dnevnikov v podatkih merilnega mesta v aplikaciji OPZP. Če ni drugače določeno, je to nosilec PK-ja iz pogodbe o dostopu. Lastnik podatkov je pravna oseba, ki je hkrati lastnik prenosnega omrežja in merilne opreme na njem. Ta oseba ima vedno dostop do vpogleda dnevnika katerega koli merilnega mesta. Za virtualna merilna mesta velja možnost dostopa glede na vlogo pravne osebe iz pogodbe o dostopu nadrejenega obračunskega merilnega mesta, ki je tisto merilno mesto, katero je bilo osnova za nastanek virtualnega merilnega mesta in na katerem se obračuna prenesena količina zemeljskega plina, ali za dobavitelja iz bilančne pogodbe.

Vnašajo se lahko le dnevniki za tista merilna mesta, ki imajo v času vpisa dnevnika tako označeno v podatku "Dnevniki odjemalcev" merilnega mesta v aplikaciji OPZP (Gema Administrator). Če je pravna oseba uporabnika lastnik podatkov, ta omejitev ne velja.

Način preračuna odjemov se prenese iz podatkov merilnega mesta iz aplikacije OPZP (nastavljivo z orodjem Gema Administrator). Preneseni dnevniki imajo status "potrjen" in so za urejanje na razpolago le aplikaciji OPZP.

2.2.4 Varnost

Za varnost dostopa do podatkov mora biti poskrbljeno na več nivojih, in sicer:

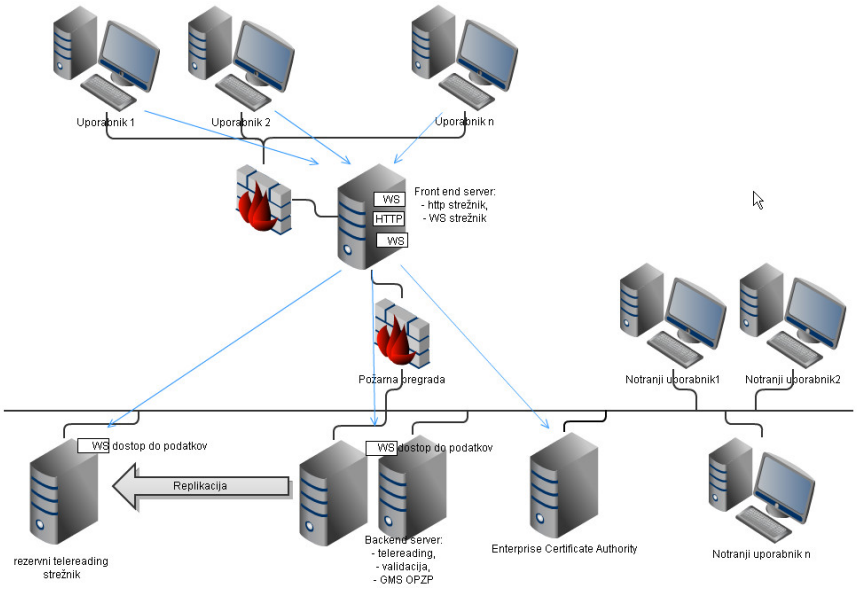
- implementira se na posebnem spletnem strežniku, ki je nameščen v DMZ coni in je zaščiten s požarno pregrado,
- izvede se varen dostop do podatkov na operativni strežnik OPZP,
- uporabi se uporabniška imena, gesla, digitalna potrdila ter kodirani protokoli (SSL),
- vgrajena mora biti vsebinska programska logika za pravice do dostopa do podatkov, vezana na veljavnosti Pogodb o dostopu.

Spletni strežnik v demilitarizirani coni DMZ je namenjen dostopu do orodja za vnos dnevnikov odjemov za koristnike, ki bodo do storitve dostopali z uporabo javnih omrežij. Komunikacija je zaščitena z uporabo varnostnih protokolov in digitalnih potrdil.

Vsem koristnikom podatkov se dodelijo digitalna potrdila ter uporabniška imena in gesla za prijavo v sistem za dostop do podatkov. Upravljanje z digitalnimi potrdili bo izvajal naročnik sam, prav tako bo zagotovil in izvajal proces izdaje. Naročnik bo uporabil obstoječo infrastrukturo za izdajo digitalnih potrdil, skladno z interno varnostno politiko.

Osnovne pravice dostopa do podatkov so vezane na veljavnosti pogodb in se določajo s programsko logiko.

Zahteve na spletni aplikacijski strežnik bodo posredovane z varovanim SSL protokolom (Secure Sockets Layer). Na spletnem strežniku bo nameščeno javno digitalno potrdilo naročnika, odjemalci pa bodo uporabljali di-



Slika 2.2: Arhitektura celotnega sistema

gitalna potrdila, podeljena s strani naročnika. Naročnik bo uredil javno strežniško digitalno potrdilo, ki se bo uporabljalo za overovitev SSL povezav.

Avtentikacija uporabnikov se izvaja z uporabo naročnikovih digitalnih potrdil, medtem ko se nastavitve avtorizacije dostopov izvajajo z uporabo administratorskega orodja GemaAdministrator. Obenem se avtorizacija veže na veljavnosti Pogodb o dostopu in bilančnih pogodb.

Poglavje 3

Opis tehnologij

3.1 Varnost na splošno

Za vzpostavitev uspešnih varnostnih mehanizmov moramo naprej razumeti, od kje izhajajo napadi, kakšen je njihov motiv in zakaj predstavljamo tarčo za napad. Naša aplikacija lahko predstavlja naključno tarčo napada, zato moramo vzpostaviti celovit varnostni sistem, ki se nenehno nadgrajuje in izboljšuje. Z varnostnimi testiranji in simuliranjem napadov lahko sami odkrijemo varnostne luknje, preden to storijo napadalci. Na takšen način aplikacijo zaščitimo, preden pride do kraje zaupnih podatkov. Napadalci so lahko različni, od naključnih vandalov, ki samo iščejo način, kako razdejati našo aplikacijo, do profesionalnih hekerjev, ki povzročijo resno finančno škodo podjetju.[4]

Pristop k varnosti v aplikacijah:[5]

- prepoznaj nevarnosti,
- zavaruj omrežje, gostitelja in aplikacijo,
- vgradi varnostne mehanizme v razvoj razvoj aplikacije.

Splošne skupine varnostnih groženj:

Kategorija	Primer grožnje / napada
Validacija vhoda	Prekoračitev pomnilnika, cross-site scripting, SQL injection
Avtentikacija	Prisluškovanje, napad z grobo silo, kraja avtentikacijskih podatkov
Avtorizacija	Zloraba privilegijev, razkritje zaupnih podatkov
Konfiguracija	Neavtoriziran dostop do administracije, zajem nešifriranih podatkov
Občutljive informacije	Dostop do shramb občutljivih podatkov
Upravljanje s sejo	Vdor v sejo, man-in-the-middle napad
Kriptografija	Slabo generirani ključi, slaba enkripcija
Manipulacija s parametri	Manipulacija s poizvedbami, manipulacija z vnosnimi polji, manipulacija s HTTP glavami
Upravljanje izjem	Razkritje informacij, distributed denial-of-service
Beleženje	Napadi brez sledi, brisanje sledi po napadih

Tabela 3.1: Tabela varnostnih groženj [5]

Varnostno testiranje išče pomanjkljivosti v varnosti aplikacije. Te pomanjkljivosti omogočajo zlonamerni kodi, da prevzame nadzor nad aplikacijo, povzroči napačno delovanje, jo onesposobi ali pridobi zaupne informacije. Varnostno testiraje bi moralo potekati skozi celoten razvoj aplikacije, da se lahko že med razvojem zakrpajo vse varnostne luknje, na žalost pa se to v večini primerov izvaja šele po koncu razvoja.

Za namen varnostnega testiranja aplikacij so razvita orodja, ki iščejo pomanjkljivosti. Skozi znane tipe napadov preverjajo varnost razvite aplikacije in poskušajo vdreti vanjo. Človek ni sposoben obdelati ogromne količine programske kode, ki nastane pri razvoju aplikacije, in preiskati vseh možnih vzorcev obnašanja. Precej lažje obvladljive in pregledne so analize, ki jih takšni programi ponudijo.

Ta orodja delimo v dve skupini - Black box in White box orodja. Black box orodja so namenjena vdiranju v aplikacijo in razkritju ranljivosti na aplikacijah, ki so že nameščene na sisteme, White box orodja pa so analitična orodja, ki preverjajo izvorno kodo aplikacije. Tako White box orodja delujejo komplementarno z Black box orodji.

Vsaki razvojni skupini mora biti varnost aplikacije glavna prioriteta. Vedno je potrebno zaščiti občutljive podatke, saj lahko odtujitev takšnih podatkov organizaciji povzroči ogromne stroške. Cilj je narediti krajo občutljivih podatkov dražjo, kot je doprinos kraje.[3, 5]

3.2 Java SE

Java je splošno namenski, objektno usmerjen programski jezik, ki temelji na razredih in je posebej zasnovan zato, da je čim bolj neodvisen od platforme. Namenjen je temu, da razvijalci razvijejo samo eno aplikacijo, ki jo potem lahko poganjajo na vseh platformah brez potrebe po spremembah ali ponovnem prevajanju kode. Javanske aplikacije se tipično prevajajo v bytecode (class datoteke), ki se izvajajo na JVM-ju, ne glede na arhitekturo računalnika ali tip operacijskega sistema. Java je popularen in široko uporabljen programski jezik predvsem pri spletnih aplikacijah tipa strežnik-odjemalec.[6, 7]

Java je bila prvotno razvita pri Sun Microsystems in leta 1995 izdana kot jedro Java platforme. Večina sintakse izhaja iz programskih jezikov C in C++, vendar so nizko nivojski mehanizmi so popolnoma drugačni. Je dokaj varna in ponuja konfigurabilno varnost, kar je omogočalo omejitve pri dostopu do datotek in omrežja. Večji spletni brskalniki so tako vključili možnost izvajanja javanskih appletov, s čimer se je njena popularnost hitro razširila.

Java je de facto standard, ki ga kontrolira Java Community Process (JCP). Ta proces omogoča zainteresiranim razvijalcem sodelovanje pri razvoju novih javanskih tehnologij. Obstaja v 3 izvedbah - poleg osnovne

različice Java SE obstajata še Java EE, ki je namenjena poslovni uporabi, in Java ME, ki je namenjena mobilnim aplikacijam. Implementacija je zapakirana v dve različni distribuciji:[6]

- Izvajalno okolje (JRE), ki vsebuje dele Jave SE, ki omogočajo izvajanje javanskih programov in so namenjeni končnemu uporabniku.
- Razvojno okolje (JDK), ki je namenjeno razvijalcem in vključuje razvojna orodja, kot so javanski prevajalnik, ki proizvede datoteke z bytecode, ki se izvajajo v izvajalnem okolju JRE, Javadoc (generator API dokumentacije, ki generira dokumentacijo v HTML formatu iz javanske kode), JAR (Java ARchive predstavlja paket javanskih class datotek, virov in metapodatkov, ki omogočajo distribucijo aplikacij) in razhroščevalnik.

Pet glavnih ciljev pri nastajanju Jave, kakšna mora biti:[6]

- enostavna, objektno-usmerjena in poznana,
- robustna in varna,
- arhitekturno neodvisna in prenosljiva,
- visoko performančna,
- interpretirana, večnitna in dinamična.

3.2.1 Varnost v Javi

Javanska platforma ponuja številne funkcije, namenjene izboljšanju varnosti v Javi. Binarna koda, ki se izvaja na virtualnem stroju, ni strojna koda za določen operacijski sistem, ampak je vmesna bytecode. Na takšen način JVM preverja bytecode, preden jo izvede, kar preprečuje, da bi programi izvedli nevarne operacije, kot je dostop do nedovoljene lokacije pomnilnika. Med izvajanjem se preverjajo tudi omejitve velikosti polj, kar pomeni, da obstaja precej manj možnosti za prekoračitev pomnilnika in ostalih nevarnosti, do

katerih lahko pride pri programih, napisanih v ostalih programskih jezikih, ki takšnih mehanizmov nimajo.

Platforma ne omogoča niti določenih nevarnih operacij, kot sta aritmetika s kazalci in nepreverjeno spreminjanje tipa. Prav tako je onemogočeno alociranje in dealociranje pomnilnika, ki ga JVM izvaja avtomatsko. Upravljanje s pomnilnikom izvaja garbage collector (GC), orodje, ki avtomatsko skrbi za sproščanje zasedenega pomnilnika. To orodje briše iz pomnilnika objekte, na katere se nihče ne sklicuje, kar dodatno pripomore k varnosti pri uporabi pomnilnika.[9]

Naslednji varnostni mehanizem, ki ščiti uporabnika, je peskovnik. Peskovnik je namenjen izvajanju kode, ki ni zaupanja vredna, zato so operacije peskovnika omejene. Program v peskovniku lahko izvaja vse operacije, vendar sta mu branje in dostop do podatkov izven peskovnika privzeto onemogočena. Namenjen je zaščiti ostalega dela sistema pred zlonamerno kodo, ki bi se lahko naložila preko medmrežja. Varnostni mehanizem v Javi prav tako omogoča podpisovanje javanskih programov, kar pove izvajalnemu okolju, da je določen program zaupanja vreden in se lahko izvaja s polnimi pooblastili. Dodatno omogoča fino nastavljanje pravic, kjer lahko za določene programe dovolimo dostop do specifičnih virov, določene programe pa popolnoma omejimo s peskovnikom. Javanski program naprej zapakiramo v datoteko tipa JAR in ga nato podpišemo.[7, 8]

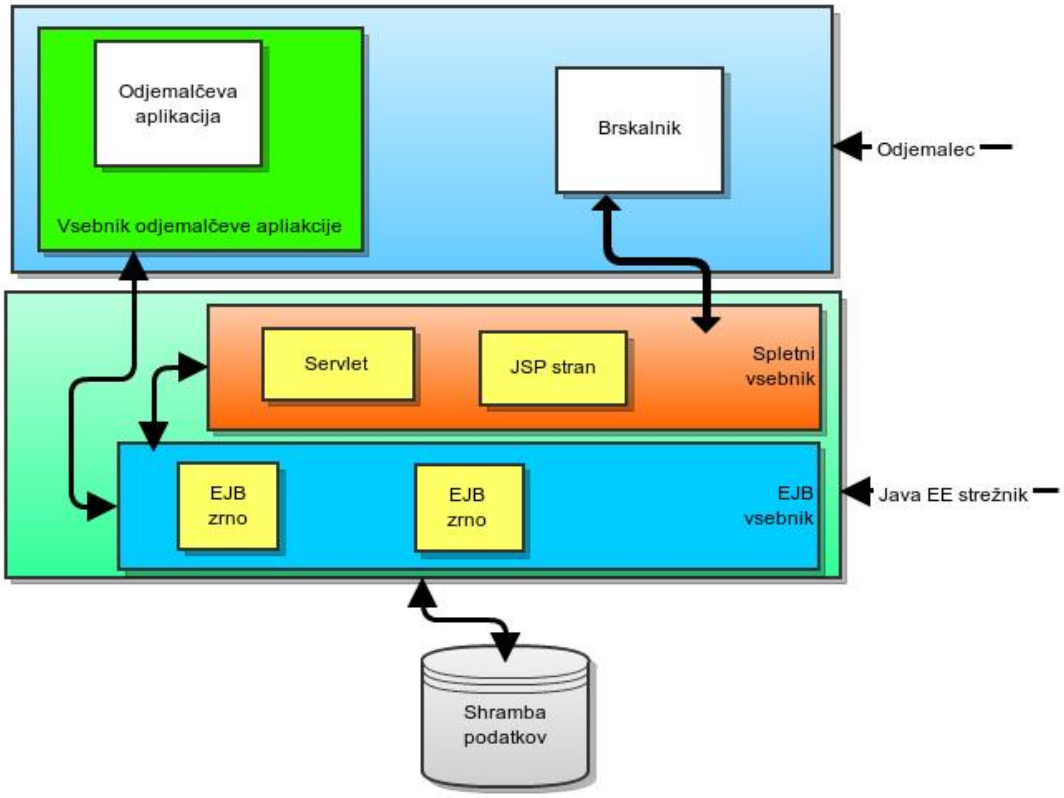
Java SE vključuje tudi kriptografsko razširitev in razširitev varnih vtičnic. Ti dve razširitvi ponujata mehanizme šifriranja, varne izmenjave ključev, zgoščevanje sporočil s preverjanjem integritete, sistem upravljanja z izmenjujočimi ključi in sistem varne povezave preko vtičnic z uporabo protokola SSL, ki deluje z uporabo kriptografske razširitve.[8]

3.3 Java EE

Spletne aplikacije vsebujejo vire, ki so dostopni širši množici in pogosto potujejo nezaščiteni po medmrežju, zato je potrebno uvesti varnostne mehanizme.

Java EE omogoča implementacijo varnostnih mehanizmov s pomočjo anotacij znotraj razredov. Ta informacije se nato uporabijo pri namestitvi aplikacije. Drugi način je deklarativno določanje varnosti, kjer s pomočjo dostopnih pravic določamo privilegije uporabnikom. To so deskriptorji, ki predpisujejo dodeljevanje pravic uporabnikom. Tretji način je programska varnost, ki je lahko precej bolj specifična od zgoraj naštetih načinov. To nam omogoča, da lahko sprejemamo varnostne odločitve za posamične akcije.

Java EE ima ogrodja, ki skrbijo za varnost. Kot je omenjeno že zgoraj, ta ogrodja skrbijo za varnost na dva načina: deklarativno in programsko.



Slika 3.1: Java EE arhitektura [11]

Deklarativno določamo varnost preko deskriptorjev XML, ki se ovrednotijo pri namestitvi aplikacije in jih obstoječa ogrodja samo uporabijo, kar

pomeni, da ne rabimo spreminjati izvorne kode. Ustrezno delujejo glede na strukturo aplikacije, modulov ali komponent. Deskriptorji definirajo strukturo posameznega dela aplikacije. Nahajajo se v zgornjem sloju vsake mape, za katero prepisujejo strukturo in varnostne nastavitve.

- Enterprise JavaBeans uporabljajo deskriptorje EJB, ki se nahajajo v META-INF/ejb-jar.xml datoteki. Deskriptorji so definirani s specifikacijo EJB 3.0 (JSR-220).
  - Atribut security-role-ref določa vsako vlogo, na katero se nanašamo v programski kodi.
  - Atribut security-role določa skupine uporabnikov in njihove privilegije.
  - Atribut method-permission določa pravice dostopa do metod.
  - Atribut run-as določa propagacijo varnostnih identitet komponente.
- Komponente za spletne storitve uporabljajo deskriptor jaxrpc-mapping-info, definiran po standardu JSR-109. Pri namestitvi določajo preslikavo funkcionalnosti med Javo in Web Service Definition Language (WSDL). V povezavi s tem delujejo tudi komponente JAX-WS 2.0, ki preslikajo anotacije iz Jave v WSDL.
- Spletne komponente uporabljajo deskriptor spletnih aplikacij web.xml. Ta datoteka definira deskriptorje po standardu Java Servlet 2.5. Določa varnostno strukturo aplikacije in pri tem definira vloge, kontrolo dostopa in zahteve za avtentikacijo. Omejitve se določajo za vsako nameščeno spletno aplikacijo posebej. Pri tem se definira omejitve dostopa do strani in navigacijske omejitve.

Primer dostopa do varnostnega konteksta klicatelja Entreprise JavaBeana:

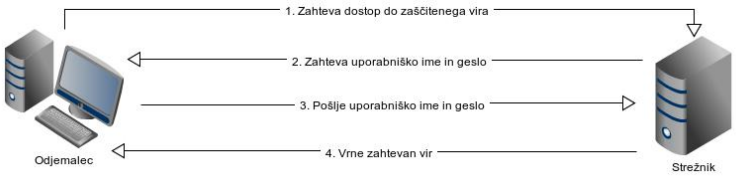
```
@Stateless
public class EmployeeServiceBean implements EmployeeService {
    @Resource SessionContext ctx;
    @PersistenceContext EntityManager em;

    public void changePhoneNumber(...) {
        ...
        // obtain caller's principals
        callerPrincipal = ctx.getCallerPrincipal();

        // obtain caller's name
        callerKey = callerPrincipal.getName();

        // use callerKey as primary key to find EmployeeRecord
        EmployeeRecord myEmployeeRecord = em.findByPrimaryKey(
            EmployeeRecord.class, callerKey);

        // update phone number
        myEmployeeRecord.setPhoneNumber(...);

        ...
    }
}
```

Izvorna koda 3.1: EJB zrno, dostop do konteksta seje in upravljalca seje

Zadnji izmed omenjenih deskriptorjev web.xml predpisuje tudi tip povezave, s katero uporabnik komunicira s strežnikom oziroma z aplikacijo, ki teče na njem. Tu lahko zahtevamo, da vsa komunikacija poteka preko varne povezave, kot je na primer HTTPS (HTTP preko SSL). Nastavitev prenosa

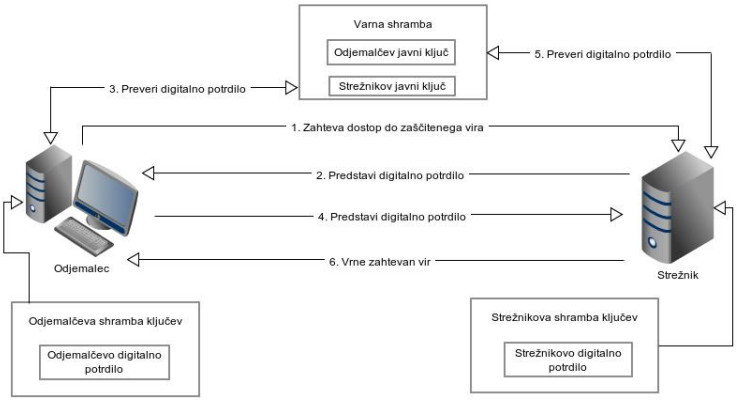
podatkov "CONFIDENTIAL" zagotavlja, da je vsa komunikacija šifrirana, in hkrati preprečuje prisluškovanje povezavi. Druga pomembna nastavitev je nastavitev avtentikacijske metode pri prijavi v aplikacijo. Če prijava v aplikacijo ni zaščitena, je napadalcu omogočeno prisluškovanje, in ker se podatki za prijavo v aplikacijo pošiljajo v čistem tekstu, je omogočena tudi prestrežba le-teh. Najvarnejša izmed vseh nastavitev je "CLIENT-CERT", ki zahteva, da se uporabnik prijavi s pomočjo digitalnega potrdila.[10]

Primer nevarne prijave:



Slika 3.2: Nevarna prijava

Primer varne prijave z uporabo digitalnih potrdil:



Slika 3.3: Varna prijava z uporabo digitalnih potrdil

3.4 Aplikacijski strežnik JBoss

Varnost je pomemben del vsake aplikacije saj varnostne luknje in izgube podatkov nas lahko drago stanejo. Varnostne luknje se lahko zlorabljajo na različne načine: neavtorizirani uporabni dobijo dostop do podatkov, prisluškovanje komunikaciji med dvema uporabniki, hekerji lahko izkoristijo varnostne luknje v omrežju ali preko aplikacijskega strežnika poganjajo ukaze na operacijskem sistemu.

Dva glavna pristopa k varnosti sta zavarovanje dostopa do podatkov v aplikaciji in zavarovanje dostopa do okolja v katerem aplikacija deluje. Hekerji lahko napadejo aplikacijo, da si pridobijo dostop do podatkov, zaganjajo zlo namerno kodo ali pa si pridobijo privilegije na operacijskem sistemu na katerem aplikacija deluje. Zato mora biti za varnost poskrbljeno z obeh vidikov. Potrebno je zavarovati tako podatke kot okolje na katerem aplikacija deluje.[13]

3.4.1 Primer uporabe varnostnih mehanizmov

V tem poglavju bosta opisana JBossov pristop k varnosti in JBossov varnostni mehanizem, imenovan JBoss SX. Za boljše razumevanje celotnega sistema in integracije med avtentikacijo, avtorizacijo in varno komunikacijo bo spodaj opisan celoten primer uporabe na primeru varne poslovne aplikacije. Primer je povzet po [13] in prirejen za aplikacijo DOO.

Kot primer imejmo uporabnika (npr. Janez), ki je energetski skrbnik podjetja, ki proizvaja železne izdelke in pri tem postopku porabi večje količine plina. Konec delovnega dne se uporabnik želi prijaviti v Java EE aplikacijo in vnesti današnji odjem plina. Podjetje, ki je lastnik aplikacije, je poskrbelo za varnost z uporabo digitalnih potrdil, ki jih je izdal zaupanja vreden overitelj digitalnih potrdil (CA). S tem njihovim uporabnikom ni potrebno skrbeti za napade tipa “phishing” ali pa “man-in-the-middle”. Janez hoče dostopati do pregleda svojih odjemov, kjer bo lahko zapisal tudi današnjega. Ker dostopa do varnega protokola HTTPS, strežnik pošlje svoje digitalno potrdilo

Janezovemu spletnemu brskalniku. Spletni brskalnik ima seznam poznanih overiteljev digitalnih potrdil, zato preveri pri enem izmed teh, če to digitalno potrdilo res pripada energetskemu podjetju, ki Janezovemu podjetju dobavlja plin. Ker eden izmed overiteljev digitalnih potrdil potrdi pristnost strežniškega digitalnega potrdila, Janezov brskalnik pošlje svoje digitalno potrdilo strani, do katere je Janez poskušal dostopati. Sedaj se lahko Janez zanese, da uporablja res pravo stran in da nihče ne prisluškuje njegovi komunikaciji s strežnikom. Prav tako se vsa komunikacija prenaša preko varnega kanala, saj uporabnik dostopa preko varnega protokola HTTPS.

Sedaj, ko Janez ve, da lahko zaupa aplikaciji, se mora še aplikacija prepričati, da lahko zaupa Janezu. URL naslov, do katerega dostopa Janez, je zavarovan in zahteva avtentikacijo. Uporabnik mora imeti pravico "GasCustomer" za dostop do izbrane strani. Še preden pa lahko Janez dostopa do izbranega naslova, njegovo zahtevo ulovi JBoss Web Server. Strežnik ugotovi, da gre za dostop do zaščitene strani, zato njegovo zahtevo posreduje modulu za varnost, da preveri, če je Janez že v aplikacijo prijavljen. Ko ugotovi, da še ni, mu strežnik vrne stran s prijavo v aplikacijo in od njega zahteva uporabniško ime in geslo.

Ko Janez vpiše svoje podatke in predloži formo, jih spletni strežnik posreduje modulu za varnost, kjer primerja vnesene podatke s tistimi, zapisanimi v bazi podatkov. Naprej gre uporabnik skozi postopek avtentikacije, s katerim strežnik preveri ali uporabniku sploh omogoči dostop do sistema. Nato gre uporabnik skozi postopek avtorizacije, kjer strežnik preveri, če ima ta uporabnik zahtevano pravico, in mu omogoči dostop do zahtevanega vira. Sedaj, ko so vsi podatki preverjeni, modul za varnost vrne nadzor spletnemu strežniku. Spletni strežnik posreduje zahtevo viru, ki je vezan na URL, da Janezu prikaže njegove odjeme. Spletna stran, ki prikazuje odjeme, je implementirana kot Servlet in zato, da se lahko stran prikaže, mora strežnik dostopati do podatkov. Servlet dostopa do metode getAllOfftakes(User u) na zrnu EJB, imenovanem OfftakeEJB, ki deluje v vsebniku EJB s katerim upravlja aplikacijski strežnik JBoss. Ker zrno EJB deluje v lokalnem

vsebniku, se preverjanje avtentikacijskih podatkov propagira samo.

Tudi zrno EJB dovoljuje dostop do svojih metod samo uporabniku s pravico GasCustomer. Vsebnik EJB dostopa do modula JBoss SX, kjer preveri, ali ima uporabnik ustrezno pravico. Ko se preverjanje zaključi, se lahko naložijo podatki iz podatkovne baze, ki se pošljejo nazaj Servletu. Slednji zgradi stran, ki jo pošlje Janezu po varnem kanalu.

3.4.2 Avtentikacija

Avtentikacija je proces, pri katerem sistem preverja identiteto uporabnika. To omogoča uporabniku, da se avtenticira na strani kot veljaven uporabnik. Uporabniki pa niso edini, ki se avtenticirajo. Tudi programi se poskušajo avtenticirati pri aplikaciji. Zato lahko govorimo o avtentikaciji uporabnika ali programa proti drugem sistemu s svojimi avtentikacijski podatki, ki so lahko uporabniško ime in geslo, digitalno potrdilo, biometrični podatki, pametne kartice, žetoni in drugi načini identifikacije.

Identifikacija je proces, pri katerem se uporabnik predstavi, medtem ko je avtentikacija proces preverjanja uporabnikovih avtentikacijskih podatkov. Če uporabimo zgornji primer, pri identifikaciji uporabnik trdi, da je Janez. Pri avtorizaciji uporabnik z digitalnim potrdilom dokaže, da je res Janez.[13]

3.4.3 Avtorizacija

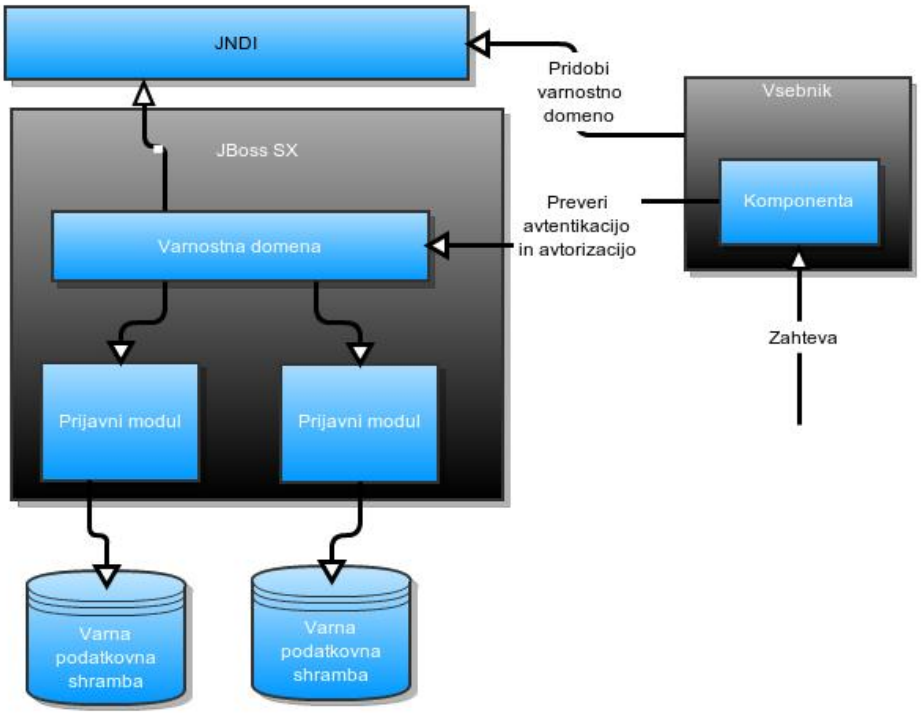
Avtorizacija je proces, pri katerem se preverja, če ima pošiljatelj zahteve ustrezne privilegije za dostop do želenega vira. Uporabnik ima lahko na primer pravico dostopa do svojih dokumentov, nima pa pravice dostopa do dokumenta, ki je v lasti drugega uporabnika. Sistem avtorizacije poteka tako, da se uporabniku dodelijo določene pravice, viru pa nabor pravic, izmed katerih je zahtevana vsaj ena za dostop do njega. Ko uporabni želi dostopati do izbranega vira, se preveri, če katera izmed uporabnikovih pravic ustreza zahtevanim pravicam za dostop. Če uporabnik izpolnjuje pogoje, se mu dostop dodeli, v nasprotnem primeru se ga zavrne.

Poglejmo primer, ko želi uporabnik Janez dostopati do administratorske strani. Za dostop do administratorske strani se zahteva pravica "Admin". Spletni strežnik pošlje zahtevo modulu JBoss SX, da preveri, ali ima Janez pravico za dostop do administratorske strani. Modul naloži vse Janezove pravice in preveri, če je katera izmed njih "Admin".[13]

3.4.4 Konfiguracija varnosti

Java EE omogoča deklarativno določanje avtorizacije za standardne tehnologije, kot so Servlet, JSP in EJB. Tak način pomeni, da se lahko izvaja avtorizacija brez pisanja dodatne kode s strani programerja. Dostopi do zrn EJB so omejeni z vlogami, definiranimi z anotacijami ali deskriptorji. Včasih pa tak način avtorizacije ni dovolj natančen. V določenih primerih potrebujemo avtorizacijo glede na kontekst. Takšen način avtorizacije je pa preveč specifičen, da bi lahko bil definiran deklarativno, zato mora biti razvit s strani programerja aplikacije in je del poslovne logike.

Vsaka Java EE komponenta ali vir ima različen mehanizem definiranja varnosti. Omogoča uporabo deskriptorjev za definicijo načina varnosti za vsak posamičen URL preko definicije vlog, ki imajo dostop do njega. Takšna definicija varnosti, kaj se naj zavaruje in na kakšen način, je definirana v vsakem deskriptorju aplikacije, Java EE pa ne predpisuje, kakšna naj bo implementacija varnosti, kje naj se shranjujejo podatki in na kakšen način se pridobivajo. Vse to je prepuščeno razvijalcem aplikacijskih strežnikov - vsak se odloči za svojo implementacijo varnosti. JBossov varnostni mehanizem se imenuje JBoss SX in je zgrajen na osnovi Java Authentication and Authorization Service (JAAS) in skrbi za vse Java EE tehnologije, ki delujejo na strežniku.[13]



Slika 3.4: Razmerja glavnih komponent varnostnega ogrodja [13]

3.4.5 Varna komunikacija

Pogosto pošiljamo podatke preko javnega omrežja, kot je na primer internet, in v tem primeru moramo poskrbeti za tri aspekte varne komunikacije: zaupnost, celovitost podatkov, celovitost vira. Zaupnost pomeni, da zavarujemo sporočilo pred vsiljivci, ki bi želeli pridobiti informacije, ki jih pošiljamo, in zagotovimo, da sporočilo dobijo samo želeni prejemniki. Za zagotavljanje zaupnosti pogosto šifriramo podatke, ki jih lahko dešifrira samo prejemnik. Celovitost podatkov pomeni, da prejemnik zaupa, da podatki med prenosom niso bili spremenjeni na nikakršen način, kar se zagotovi s šifriranjem podatkov. Celovitost vira zagotavlja, da prejemnik zaupa, da sporočila res prihajajo od pravega pošiljatelja in ne od nekoga drugega, ki se predstavlja kot pošiljatelj. Celovitost vira lahko zagotovimo z uporabo digitalnih

potrdil, izdanih s strani overitelja digitalnih potrdil. Tako vsak, ki si hoče pridobit zaupanje drugih udeležencev komunikacije, pridobi digitalno potrdilo in ga pošlje drugim udeležencem, ti pa lahko njegovo pristnost preverijo pri overitelju digitalnih potrdil.

Za zagotavljanje teh aspektov varnosti uporabljamo protokole, kot so Transport Layer Security (TLS) ali Secure Sockets Layer (SSL). Ti protokoli uporabljajo kombinacijo javnih in zasebnih ključev.[13]

Avtentikacija odjemalca na podlagi digitalnega potrdila

V komunikaciji SSL strežnik zahteva uporabo digitalnih potrdil. Najpogosteje ima strežnik digitalno potrdilo, medtem ko ga odjemalec ne potrebuje. Avtentikacija strežnika pomeni, da strežnik in odjemalec uporabljata varno povezavo SSL, vendar ima digitalno potrdilo samo strežnik. V tem primeru, ko odjemalec dostopa do strežnika, preveri strežnikovo digitalno potrdilo med rokovanjem. SSL pa je sposoben tudi avtentikacije odjemalca. To se zgodi pri obojestranski avtentikaciji, ko odjemalec dostopa do strežnika in pri rokovanju izmenično preverita digitalni potrdili. Pri aplikacijskem strežniku JBoss se avtentikacija SSL lahko uporablja pri dostopu do spletnega strežnika ali pa do modula, ki skrbi za zrna EJB. Obojestranska avtentikacija ni prav pogosta, saj bi to pomenilo, da bi morali vsi uporabniki spletnih storitev pridobiti in namestiti digitalno potrdilo v svoje brskalnike. Uporablja se le v primerih, ko spletna aplikacija zahteva višjo stopnjo varnosti, ko dostopamo do zaupnih podatkov, pridobitev katerih bi s strani neavtorizirane osebe lahko pomenila resno škodo. Vedno lahko izvedemo avtentikacijo odjemalca z uporabo gesla, če pa strežnik zahteva višjo stopnjo varnosti in zaupanja, uporabi digitalna potrdila, kar hkrati zagotavlja, da se vsi podatki šifrirajo, preden se pošljejo po omrežju.[13]

SSL v varnostni domeni

JBoss ima vgrajeno podporo za uporabo protokola SSL preko HTTP vtičnika, kar omogoča zelo enostavno konfiguracijo varne povezave SSL, vendar, če

želimo avtenticirati uporabnike na podlagi digitalnega potrdila, je potrebno definirati varnostno domeno, ki se zaveda protokola SSL. Ta proces zahteva definicijo MBeana, ki ga predpisuje JassSecurityDomain, ki kaže na varno shrambo digitalnih potrdil.

```
<server>
  <mbean code="org.jboss.security.plugins.JaasSecurityDomain"
      name="jboss.web:service=MySecurityDomain">
    <constructor>
      <arg type="java.lang.String" value="my-security-domain"/>
    </constructor>
    <attribute name="KeyStoreURL">
    ${jboss.server.home.dir}/conf/server.truststore</attribute>
    <attribute name="KeyStorePass">servercert</attribute>
    <depends>jboss.security:service=JaasSecurityManager</
       depends>
  </mbean>
</server>
```

Izvorna koda 3.2: Konfiguracija varnostne domene

Primer takšne konfiguracije varnostne domene je naveden v izvorni kodi 3.2. Definirali smo ji ime “my-security-domain”, kar jo pri namestitvi aplikacije poveže v JNDI kontekst z imenom “java:/jaas/my-security-domain”. Dodatno moramo definirati še pot do shrambe in ključa shrambe. Če želimo našo varnostno domeno uporabljati tudi za avtentikacijo in avtorizacijo, potem moramo v prijavnem modulu definirati našo domeno z naslednjim ukazom:[13]

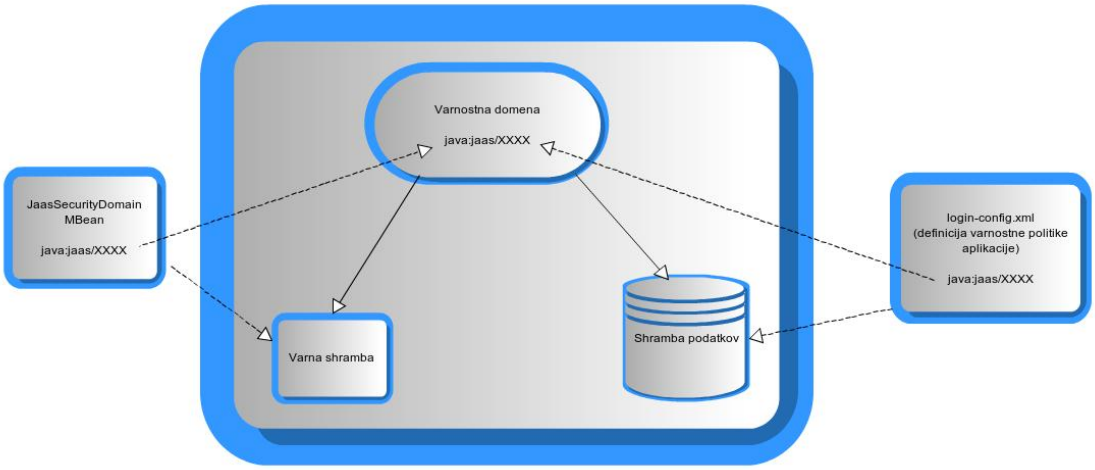
```
1 <application-policy-name="my-security-domain">
2 ...
3 </application-policy-name>
```

Izvorna koda 3.3: Definicija varnostne domene

3.4.6 Konfiguracija prijavnega modula

Privzeto, spletne aplikacije niso varne. Če spletne aplikacije ne zavarujemo, lahko vsak dostopa do vsakega URL naslova nameščene aplikacije. Zgoraj smo govorili o varnosti strežnika JBoss, konfiguraciji datoteke server.xml, ki omogoča konfiguracijo varnosti na strežniku, v naslednjem odstavku pa se bomo posvetili konkretni varnosti na spletni aplikaciji.

Vsaka spletna aplikacija ima v mapi WEB-INF standardno opisno datoteko web.xml in JBoss specifično datoteko jboss-web.xml. Ti dve datoteki predpisujeta varnost v spletni aplikaciji, omejujeta dostop do URL naslovov aplikacije in se povezujeta z varnostno domeno za zagotavljanje varnosti znotraj aplikacije same. Datoteka login-config.xml vsebuje podatke varnostne domene in pri zahtevi preveri podane podatke s tistimi, ki jih domena hrani v svoji varni shrambi. V konfiguraciji datoteke web.xml specificiramo varnost za vsak URL vzorec posebej in HTTP metodo, pri kateri želimo zagotoviti varnost. Vzorce URL naslovov definiramo tako, da definiramo pot naslova in podaljšek datoteke. Z znakom "/" pa definiramo privzet Servlet aplikacije. Ko smo definirali, kateri URL naslovi so varni, moramo definirati tudi vloge, ki imajo pravico za dostop do teh naslovov. Navesti moramo imena vlog, ki imajo pravico za dostop do teh naslovov in samo uporabniki s temi vlogami lahko dostopajo do njih. Naslednja spremenljivka, ki jo lahko nastavimo, je način prenosa informacij. Tukaj lahko zahtevamo, da se vse informacije pretakajo šifrirano. Seveda mora biti vsak uporabnik predhodno avtenticiran za celotno aplikacijo z uporabni prijavnega modula.



Slika 3.5: Povezava med prijavnim modulom in JaasSecurityDomain [12]

Datoteka web.xml definira samo kateri URL naslovi so varovani, ne pa tega, kako jih varujemo. To vlogo ima datoteka login-config.xml, ki nato kaže na jboss-web.xml.[13]

3.5 Spletne storitve

Z uvedbo Jave SE 5.0 in njenih anotacij za podporo spletnim storitvam je postala izdelava in uporaba spletni storitev zelo preprosta. V tem poglavju se bomo osredotočili na spletne storitve, ki jih uporablja JBoss, in sicer na Java API za spletne storitve, ki bazirajo na XML dokumentih (JAX-WS).[13]

3.5.1 Kaj so spletne storitve

Rečemo lahko, da so spletne storitve klic oddaljene metode (RMI) preko protokola HTTP z uporabo transportnega mehanizma, ki bazira na dokumentih XML. Spletne storitve niso nobena revolucija, vendar nam omogočajo integracijo raznolikih sistemov med seboj.

Spletne storitve nam omogočajo preprosto izmenjavo podatkov med dvema različnima sistemoma. Strani, ki si izmenjujejo podatke, se morajo samo do-

govoriti glede standarda, po kateremu si te podatke izmenjujejo. Uporaba XML-a kot privzetega standarda je bistveno olajšala celoten sistem komunikacije.

Druga prednost, ki jo ponujajo, je, da delujejo preko protokola HTTP, kar omogoča uporabo na sistemih, ki so zaščiteni s požarnimi zidov. Vrata, na katerih deluje ta protokol, so tipično odprta navzven, vrata, katera uporablja na primer RMI, so pa tipično zaprta.[13]

3.5.2 Zagotavljanje varnosti v spletnih storitvah

Avtorizacija na spletnih storitvah se deli na dva dela, glede na to, ali je nosilec spletne storitve POJO ali EJB.

Pri spletnih storitvah, ki temeljijo na objektih POJO, se varnost zagotavlja na isti način kot pri Servletih, in sicer z definiranjem varnostnih omejitev v datoteki web.xml in jboss-web.xml. Pri dostopu do takih spletnih storitev je vse, kar potrebujemo, avtentikacija. Ustvarimo JAX-WS Factory, kateri navedemo uporabniško ime in geslo in ustvarimo posrednika (proxy) do spletne storitve. V izvorni kodi 3.4 je primer takšnega odjemalca.

```
1 JaxWsProxyFactoryBean factory = new JaxWsProxyFactoryBean();
2 factory.getInInterceptors().add(new LoggingInInterceptor());
3 factory.getOutInterceprots().add(new LogginOutInterceptor());
4 factory.setServiceClass(POJOWebService.class);
5 factory.setAddress("http://localhost:8080/pojoService");
6 factory.setUsername("admin");
7 factory.setPassword("passwd");
8 POJOWebService client = (POJOWebService)factory.create();
9 client.doSomething();
```

Izvorna koda 3.4: Odjemalec spletnih storitev

Pri spletnih storitvah, ki temeljijo na tehnologiji EJB, je konfiguracija malo drugačna, ker varnostna domena ni specificirana s pomočjo spletnih

deskriptorjev, ampak jo moramo definirati z uporabo anotacij na zrnih EJB.

```
1 @Stateless
2 @WebService(targetNamespace= "http://www.mydomain.com/", serviceName
    ="SecureEJBService")
3 @WebContext(authMethod="CLIENT-CERT", sercureWSDLAccess = false,
    transportGuarantee="CONFIDENTIAL")
4 @SecurityDomain(value = "mysecuritydomain")
5 public class SecureEJB {
6 ...
7 }
```

Izvorna koda 3.5: Spletna storitev, ki temelji na zrnu EJB

V zgornjem primeru smo uporabili anotacijo @WebContent in nastavili parametre. Parameter authMethod nam omogoča način avtentikacije uporabnika. Vrednost "CLIENT-CERT" zahteva, da se uporabnik predstavi z digitalnim potrdilom. TransportGuarantee definira, na kakšen način se prenašajo podatki. Privzeta vrednost ne uporablja protokola SSL in prenaša podatke v navadnem tekstu, vrednost "CONFIDENTIAL" iz zgornjega primera pa zahteva šifriranje podatkov pred prenosom in hkrati zagotavlja, da se podatki med prenosom ne morejo spreminjati. Ta način prenosa seveda uporablja protokol SSL. Vrednost secureWSDLAccess predpisuje, ali lahko uporabniki javno dostopajo do datoteke WSDL ali za to zahtevajo avtentikacijo. Pri vrednostni false lahko vsak uporabnik vidi, katere metode so javno izpostavljene in kakšne parametre imajo. Za dostop do teh metod se mora še vedno avtenticirati na predpisan način.

Varnost na spletnih storitvah vključuje avtentikacijo, avtorizacijo in enkripcijo. Spletne storitve privzeto omogočajo dostop vsem. Medtem ko je za spletne storitve, ki so dostopne samo v zaprtem omrežju, ali za takšne, ki ponujajo javno dostopne podatke, to primerno, pa je za takšne, ki ponujajo zaupne podatke, to slaba rešitev.

JBoss privzeto uporablja JBossWS varnostno domeno, kar lahko preverimo in nastavimo v datoteki server/SEVER_INSTANCE/conf/login-config.xml. Tu lahko nastavimo druge načine preverjanja, kot so uporaba LDAP-a ali podatkovne baze. Za potrebe primera ostanimo pri varnostni domeni JBossWS. Ta varnostna domena ima specificirane uporabnike v datoteki server/SERVER_INSTANCE/conf/props/jboss-users.properties in vloge v datoteki jbossws-roles.xml. Konfiguracija poteka tako, da v datoteki navedemo uporabniško ime in geslo, v drugi datoteki pa uporabniška imena povežemo z vlogami, katere uporabniku omogočajo dostop do virov.

Spletne storitve, ki bazirajo na objektih POJO, pakiramo v datoteko WAR, zato uporabljajo isti način konfiguracije varnosti kot Servleti ali datoteke JSP. Z deskriptorjem predpišemo, kateri uporabniki imajo dostop do katere metode spletne storitve.

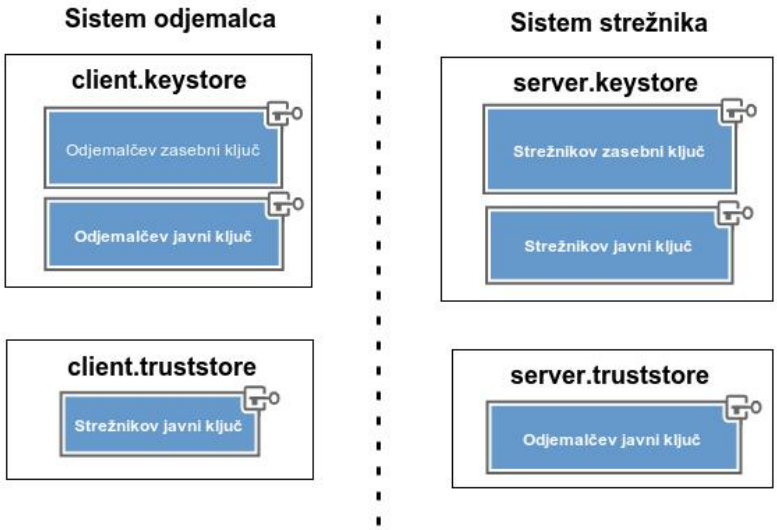
Na primeru izvorne kode 3.6 vidimo, da naša spletna storitev uporablja kontekst /tax in tega želimo zavarovati. To vrednost enačimo z elementom urlPattern pri uporabi anotacije @WebContext. Definirali smo, da mora uporabnik, ki je v varnostni domeni JBossWS, imeti vlogo “merchant” in za prijavo mu zadoščata uporabniško ime in geslo, torej ne potrebuje digitalnega potrdila. Navedli smo tudi, da varujemo samo zahteve tipa POST. JBossov JAX-WS API nima mehanizma, ki bi omogočal specifikacijo uporabniškega imena in gesla pri dostopu do WSDL-ja, kateri uporablja metodo GET. Na tak način zavarujemo metode spletne storitve in uporabniku še vedno omogočimo dostop do WSDL-ja.[13]

```
<web-app>
  ...
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>Secure Sales Tax</web-resource-name>
      <url-pattern>/tax</url-pattern>
      <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
      <role-name>merchant</role-name>
    </auth-constraint>
  </security-constraint>
  <login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>JBossWS</realm-name>
  </login-config>
  <security-role>
    <role-name>merchant</role-name>
  </security-role>
</web-app>
```

Izvorna koda 3.6: Varnost v spletnih aplikacijah je definirana v web.xml datoteki

3.5.3 Uporaba digitalnih potrdil pri zagotavljanju varne komunikacije med odjemalcem in spletno storitvijo

Zgoraj smo se osredotočili na varnost pri dostopu do spletnih storitev. Pogledali smo način, kako zaščitimo spletne storitve, ki temeljijo na objektih POJO ali na tehnologiji EJB. Za popolno varnost moramo poskrbeti tudi za



Slika 3.6: Shramba ključev in varna shramba pri komunikaciji strežnik - odjemalec [14]

prenos podatkov, poskrbeti moramo, da so podatki pred transportom kriptirani. To lahko dosežemo na dva načina, in sicer z uporabo protokola SSL za prenos preko HTTPS ali uporabimo enkripcijo JAX-WS. Prvi način je zelo podoben uporabi protokola HTTPS pri spletnih aplikacijah, pri drugem pa implementacija JAX-WS na obeh straneh poskrbi za enkripcijo.

Pošiljanje zahteve in odgovor na zahtevo sta sestavljena iz dveh sporočil in obe sporočili morata biti kriptirani. Seveda lahko kriptiramo obe strani z istim digitalnim potrdilom, vendar si v produkciji nikoli ne želimo izpostavljati našega zasebnega digitalnega potrdila, saj ga želimo obdržati čim bolj skritega. V te namene želimo za strežnik in vsakega odjemalca izdelati svoje digitalno potrdilo.

V primeru enega strežnika in enega odjemalca moramo generirati dve shrambi ključev (keystore) in dve varni shrambi (truststore). Shramba ključev vsebuje lasten zasebni in javni ključ, medtem ko varne shrambe vsebujejo javni ključ druge strani.

Na sliki 3.6 imamo prikazan primer takšnega odnosa strežnik - odjemalec. V Javi imamo orodje keytool, ki nam pomaga pri vzpostavitvi takšnega sistema uporabe digitalnih potrdil. Z naslednjimi ukazi (izvorna koda 3.7) lahko vzpostavimo takšno konfiguracijo:

```
1 keytool -genkey -alias server -keyalg RSA -keystore server.
    keystore
2 keytool -genkey -alias client -keyalg RSA -keystore client.
    keystore
3 keytool -export -alias server -keystore server.keystore -file
    server_pub.key
4 keytool -export -alias client -keystore client.keystore -file
    client_pub.key
5 keytool -import -alias client -keystore server.truststore -file
     client_pub.key
6 keytool -import -alias server -keystore client.truststore -file
     server_pub.key
```

Izvorna koda 3.7: Generiranje ključev

Ko nam je orodje keytool zgeneriralo digitalna potrdila in pripadajoče shrambe, moramo le-te povezati z aplikacijskim strežnikom. Konfigurirati je potrebno datoteko jboss-wsse-server.xml, v kateri bomo identificirali shrambo ključev in varno shrambo. Za spletne storitve, ki bazirajo na objektih POJO, je potrebno to datoteko postaviti v mapo WEB-INF, medtem ko jo za spletne storitve, ki bazirajo na tehnologiji EJB, postavimo v mapo META-INF. V datoteki z uporabo elementov <key-store-file>, <key-store-type>, <key-store-password> in enakih trust-store elementov definiramo podatke o naših shrambah. <key-store-file> zahteva pot do shrambe, relativno na datoteko WAR, <key-store-password> je geslo, ki smo ga uporabili pri generiranju shramb z orodjem keytool, medtem ko je <key-store-type> v našem primeru vedno JKS. Navesti moramo tudi geslo strežniškega digitalnega potrdila z

uporabo elementa <key-password>. Ta nam omogoča dostop do strežniškega digitalnega potrdila v shrambi ključev.

Na koncu moramo še na izpostavljenem vmesniku spletne storitve deklarirati anotacijo @EndpointConfig, s katero navajamo, da želimo uporabiti WS-Security. Privzete varnostne nastavitve, ki se že nahajajo v JBossovi konfiguracijski datoteki server/SERVER_INSTANCE/deploy/jbossws.sar/META-INF/standard-jaxws-endpoint-config.xml, se imenujejo "Standard WSSecurity Endpoint" in jih definiramo v anotaciji @EndpointConfig na parametru "configName". Uporabnik lahko po želji definira druge deskriptorje, ki uporabljajo druge upravljavce varnostnih mehanizmov.

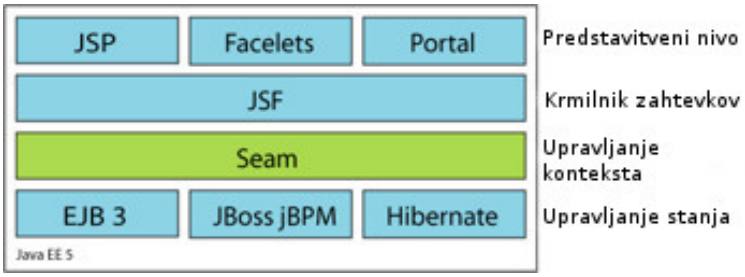
Za dostop do spletne storitve in uporabo kriptiranih sporočil moramo konfigurirati tudi odjemalca. Sam odjemalec ne potrebuje nobene dodatne kode, potrebuje le konfiguracijsko datoteko jboss-wsse-client.xml, po vzoru tiste s strežnika. Vse, kar moramo spremeniti, je, da navedemo poti do shramb ključev na strani odjemalca, geslo odjemalčevega digitalnega potrdila ter da datoteko postavimo v mapo META-INF. Enako konfiguracijo lahko izvedemo tudi med izvajanjem odjemalca s sistemskimi nastavitvami, vendar, če smo konfiguracijo navedli tako s konfiguracijsko datoteko kot s sistemskimi nastavitvami, bo imela datoteka prednost.

WS-Security omogoča podpisovanje sporočil. Ta pristop nam omogoča še en način avtentikacije uporabnika. Za podpis sporočila pošiljatelj uporabi svoj zasebni ključ, prejemnik pa uporabi pošiljateljev javni ključ, da preveri pošiljateljevo identiteto. Za takšen sistem morata imeti oba, strežnik in odjemalec, v svojih varnih shrambah svoj in nasproten javni ključ. Konfiguracijo varnostni moramo dopolniti tako, da navedemo, katere ključe bomo uporabili za popisovanje sporočil.[13]

3.6 Ogrodje Seam

Seam je ogrodje za izgradnjo aplikacij, ki temeljijo na Java EE. Določa enoten model komponent za celotno poslovno logiko v aplikaciji. Seamove komponente imajo stanje, ki se hrani v kontekstu, in to stanje ohranjajo čez celoten tok uporabnikove interakcij s spletno aplikacijo. Seam združuje tudi sloje aplikacije. Ne zahteva več stroge ločitve predstavitvenega nivoja in nivoja poslovne logike. Uporabniku omogoča, da sam izbere naravno ločitev komponent med seboj in ni več omejen s tem, kar ogrodje zahteva od njega. V določenih primerih takšna delitev postane prisiljena in nenaravna, omejuje razvijalca aplikacije, Seam pa odpravlja to pomanjkljivost, ki jo vidimo pri nekaterih drugih podobnih ogrodjih.

Ob prihodu Jave EE 5 sta prišli dve novi tehnologiji, JSF in EJB3. EJB3 je model komponent za upravljanje s poslovno logiko in persistentnimi objekti, JSF pa komponenti model za uporabo na predstavitveni plasti. Gre za zelo dobra modela, vendar je to, kar manjka, integracija med njima. Tu nastopi Seam in ta dva modela združi. Programerja aplikacije reši pisanja dodatne kode za potrebe integracije teh dveh modelov in mu omogoči, da svoj čas usmeri v razvoj poslovne logike. Tako vsako zrno EJB ali objekt POJO postane Seamova komponenta samo z uporabo anotacije @Name.[15, 16]



Slika 3.7: Sklad JavaEE tehnologij pri spletnih aplikacijah in vloga Seama [17]

3.6.1 Varnost v Seamu

Vloga varnostne plasti je preprečevanje dostopa neavtoriziranim uporabnikom do občutljivih področij aplikacije. Glede na pomembnost te plasti mora biti enostavna za konfiguracijo ter uporabo in integrirana v jedro aplikacije. Pomembno je, da pri poenostavitvi konfiguracije in uporabe ne izgubimo na zmogljivosti in skalabilnosti. In za vse to so pri Seamu poskrbeli.[15]

Avtentikacija in avtorizacija

Uporabnik z avtentikacijo pridobi identiteto v aplikaciji. Anonimen uporabnik ima sedaj svojo identiteto, katere se aplikacija zaveda. Med Seamovo avtentikacijo le-ta združi objekti iz JAAS-ovega API-ja z uporabniško sejo. Tako uporabnik obdrži svojo identiteto za celoten čas trajanja seje.

Takšna uporabnikova identiteta nosi dva tipa podatkov. Prvi podatek je uporabniško ime, s katerim se je uporabnik avtenticiral, drugi podatek je pa seznam vlog, katere ima uporabnik. Te vloge predstavljajo pravice, s katerimi uporabnik dostopa do virov aplikacije. Vsi objekti so instance objektov JAAS API-ja, iz katerega Seam pridobi vse podatke pri prijavi uporabnika v aplikacijo.[15]

Avtentikacijska metoda in vloga JAAS

Seam se v ozadju za svoje delovanje zanaša na JAAS, vendar pa uporabnik Seama ne pride v direkten stik z JAAS-om, saj Seam ponuja svoj nivo varnosti, preko katerega uporabnik izvede vsa potrebna varnostna preverjanja, iz katerih pridobi vse potrebne informacije.

Za implementacijo avtentikacije moramo navesti, kje se nahajajo uporabniški računi. Uporabnik ima pri tem popolnoma svobodno izbiro, lahko pa si pomaga z nastavki za branje podatkov iz zbirke podatkov ali LDAP-a. Tako je Seamova avtentikacija omogočena v treh korakih:

- Omogočimo avtentikacijo s konfiguracijo avtentikacijske metode.

- Preverimo uporabnikovo uporabniško ime in geslo v avtentikacijski metodi.
- Ustvarimo prijavno formo.

Avtentikacija je privzeto omogočena, vendar dokler Seamu ne povemo, kako naj uporablja avtentikacijo, prijava kakršnega koli uporabnika ni mogoča. Za zadovoljitev te zahteve, Seam zahteva metodo brez parametrov, ki vrača vrednost boolean in ki je dostopna iz EL-ja (expression language). Vse drugo postori Seam z integracijo metode v JAAS-u. Tako JAAS pokliče našo metodo in preko SeamLoginModula dobimo identiteto, preko katere lahko izvajamo celotno avtorizacijo uporabnika. Identiteta živi v kontekstu komponente in vsebuje referenco na našo avtentikacijsko metodo. Na tak način dodaten sloj poskrbi, da se vse avtentikacijske zahteve izvajajo v ozadju preko varnih in preverjenih mehanizmov. Avtentikacijska metoda se lahko nahaja v katerem koli razredu. Spodaj je naveden čisto preprost in delujoč primer takšne metode v izvorni kodi 3.8:

```
1 @Name(value = "authenticationManager")
2 public class AuthenticationManager {
3   public boolean authenticate() {
4     return true;
5   }
6 }
```

Izvorna koda 3.8: Prazna avtentikacijska metoda

Seveda takšen preprost primer ni primeren za prehod v produkcijo. Resna avtentikacijska metoda mora preveriti uporabnikovo uporabniško ime in geslo. Glede na to, da metoda ne prejme nobenih parametrov, moramo pridobiti podatke o uporabniku na drug način. To naredimo tako, da iz konteksta seje pridobimo identiteto. Takšna avtentikacijska metoda bi izgledala kot v izvorni kodi 3.9.

```
@Name(value = "authenticationManager")
public class AuthenticationManager {
  @In private Identity identity;
  @In private Session session;
  @In private PasswordManager passwordManager;

  public boolean authenticate() {
    Query query = session.createQuery("from Member m where m.username
        = :username");
    query.setParameter("username", indentity.getUsername())
    Member member = (Member)query.uniqeResult();
    if (passwordManager.validatePassword(identity.getPassword(),
       member)) {
     identity.addRole("member");
      return true;
    } else {
      return false;
    }
  }
}
```

Izvorna koda 3.9: Seamova komponenta za avtentikacjo. V authenticate() metodi se izvaja avtentikacija uporabnika.

Kot vidimo, nosi objekt Identity informacijo o uporabniškem imenu, geslu in seznam pravic, ki jih uporabnik ima. Med procesom avtentikacije nato JAAS prenese vse te informacije na svoje objekte. S poizvedbo pridobimo iz shrambe podatkov uporabnika s podanim uporabniškim imenom in nato primerjamo gesli, če se ujemata. Če se gesli ujemata, Seam vrne nadzor JAAS-u, ki izvede prijavno proceduro, v nasprotnem primeru uporabnika vrnemo na prijavno stran.

V tretji fazi moramo pripraviti prijavno formo. Prijavna forma ustreza Servlet specifikaciji, ki nam omogoča uporabo preproste forme JSF. V formi

povežemo dve vrednosti #{indentity.username} in #{identity.password}, preko katerih dobimo vrednosti v avtentikacijski metodi na objektu identity. Vse skupaj povežemo na akcijo #{identity.login}, kar sproži izvajanje avtentikacijske metode. Primer takšne prijavne forme je v izvorni kodi 3.10.

```
<h:form id="login">
  <h:panelGrid columns="2">
    <h:outputLabel for="username" vaule="Username"/>
   <h:inputText id="username" value="#{identity.username}"/>
   <h:outputLabel for="password" value="Password"/>
   <h:inputSecret id="password" value="#{identity.password}"/>
  </h:panelGrid>
  <div class="actionButton">
   <h:commbaButton value="Login" action="#{identity.login}"/>
  </div>
</h:form>
```

Izvorna koda 3.10: Prijavna forma v JSF z uporabo Seamove komponente Identity

Pri vsakem poskusu prijave se geslo pobriše. Ob neuspešni prijavi metoda login() vrne vrednost null, kar povzroči ponovno nalaganje prijavne strani. Ob uspešni prijavi metoda vrne vrednost loggedIn, preko katere v navigacijskih pravilih preusmerimo uporabnika v aplikacijo.

```
<navigation from-action="#{indentity.login}">
  <rule if-outcome="loggedIn">
    <redirect view-id="/home.xhtml"/>
  </rule>
</navigation>
```

Izvorna koda 3.11: Navigacijsko pravilo

V Seamu je celoten postopek avtentikacije skrit pred nizkonivojskim JAAS-om in ga lahko izvedemo v 3 preprostih korakih.[15]

3.6.2 Integracija varnostnih mehanizmov

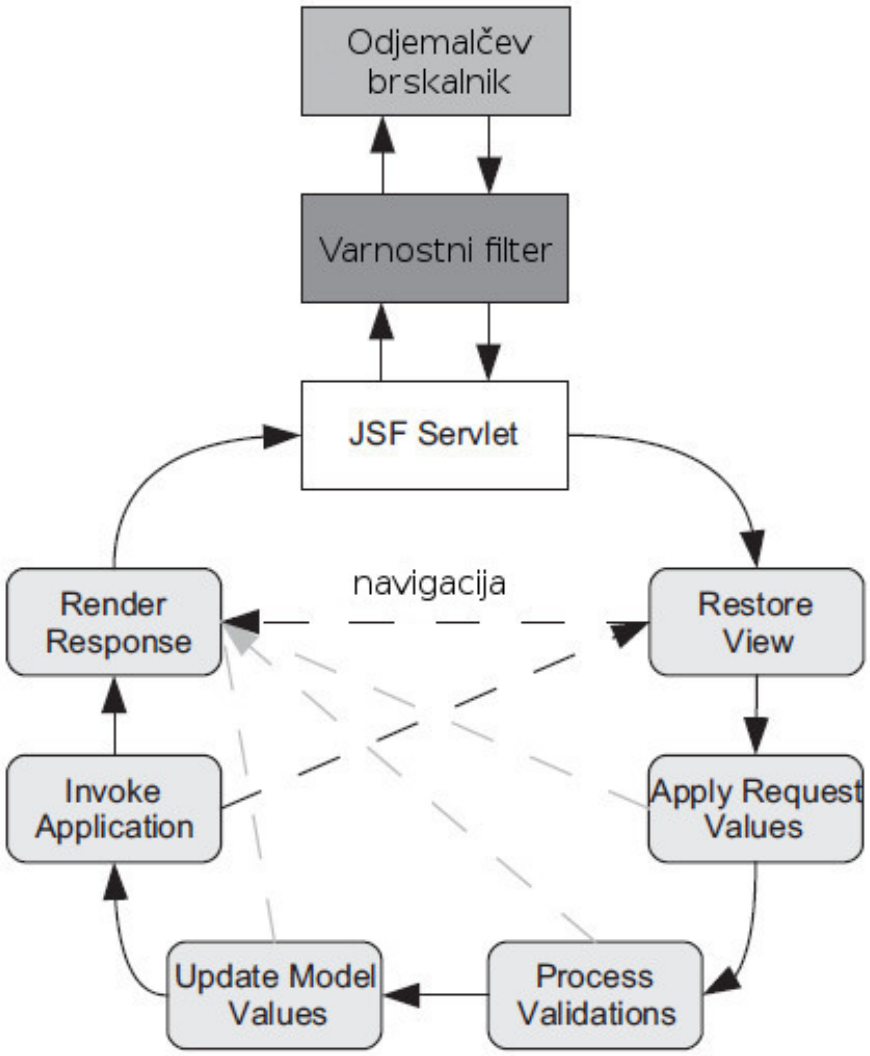
Najbolj pogost način zagotavljanja varnosti v spletnih aplikacijah je varnost na nivoju strani. Tudi pri zagotavljanju varnosti na komponentah še vedno potrebujemo dostop do spletne strani, da lahko uporabnika preusmerimo na prijavno stran, če ni prijavljen, ali pa mu prikažemo napako, če nima dostopa do želenega vira.

Največja težava JSF-ja je, da ne vsebuje varnostnih mehanizmov. Sam dizajn je osnovan tako, da so ta mehanizem izpustili, saj mislijo, da je to v domeni kakšnega drugega nivoja, kot je na primer modul za upravljanje z zrni EJB ali Servletov filter. Seam z združevanjem teh tehnologij poskrbi za varnost na obeh nivojih, tako na nivoju strani kot na nivoju komponent. Lahko bi celo rekli, da je varnost v Seamu največji prispevek k Javi EE.

Na prvi pogled se zdi, da bi lahko bil Servletov filter popolna implementacija varnosti na nivoju strani. Filter ujame zahtevo in lahko se odloči, ali uporabnika spusti na želeno stran ali ga preusmeri na neko drugo stran. Težava je v tem, da ta filter deluje na previsokem nivoju in se ne zaveda dogajanja znotraj JSF-jevega življenjskega cikla. Na primeru bomo opisali, kako lahko naletimo na težave zaradi zgoraj omenjenih pomanjkljivosti.

Ko zahtevamo dostop do URL naslova, ogrodje JSF najprej izriše podlogo strani, ki ustreza izbranemu identifikatorju. Recimo, da smo v filtru nastavili omejitev dostopa do URL naslova na podlagi nekega pravila. Stran ima formo JSF z ukazom grafičnega vmesnika, kateri pokliče neko akcijo. Ko uporabnik klikne na gumb, se sproži zahtevek do istega URL naslova in zgodi se isto preverjanje kot pri dostopu do trenutne strani. Po izvedeni akciji, JSF pokliče navigacijsko metodo, ki lahko vrne drugo identifikacijo, kar povzroči izris druge strani. Varnostni modul se te zamenjave strani ne zaveda in tako ne more preveriti, ali je uporabnik upravičen do nove zahtevane strani. Takšen primer prikazuje slika 3.8, na kateri lahko vidimo tudi, kako pri tem obidemo

varnostni filter.



Slika 3.8: JSF-jev življenjski cikel in prikaz načina, kako obiti varnostni filter v ogrodju JSF [15]

Seveda to mapiranje z uporabo filtra deluje samo, če uporabljamo Servletov razpošiljatelj zahtev. Faceleti uporabljajo svoj mehanizem in v tem primeru zahteve ponovno obidejo varnostni filter.

Še ena pomanjkljivost takšnega filtra je, da nima dostopa do aplikacij-

skega konteksta. Da se lahko pravilno odločimo, ali uporabniku dovolimo dostop do izbranega vira, bi potrebovali tudi informacijo, od kje uporabnik prihaja, kaj počne in kakšno je trenutno stanje seje. Edini način, da bi lahko dobili vse te informacije, je, da bi bil varnostni modul integriran v JSF. In prav to težavo Seam rešuje. Namesto, da se preverjanje izvede pred in po vsaki zahtevi, Seam uporablja varnost na nivoju strani in izvaja preverjanje med dvema fazama cikla JSF, in sicer "Restore View" in "Render Response". V tem primeru se lahko omejitve nanašajo direktno na identifikacijo zahtevane strani in ne več na URL naslov. V primeru, ko uporabnik nima dostopa do takšnega vira, ga Seam preusmeri na stran, ki prikazuje napako, ali ponudi uporabniku možnost avtentikacije.[15]

Preverjanje uporabnikovih pravic

V primeru, da skuša neavtenticiran uporabnik dostopati do zaščitenega vira, ga Seam preusmeri na prijavno stran, kjer se identificira. Ko je uporabnik identificiran, se izvede proces avtorizacije in se preveri, če ima uporabnik pravico dostopa do želenega vira. Seveda je potrebno Seamu predhodno povedati, kam naj preusmeri neavtenticiranega uporabnika. Za izvedbo avtentikacije obstajata dva načina. Prvi je, da uporabnik sam dostopa do prijavne strani, vendar se v praksi izkaže, da večina uporabnikov direktno dostopa do želenega vira mimo prijavne strani. V Seamu lahko zaščitimo stran tako, da od neavtenticiranih uporabnikov zahtevamo, da se najprej prijavijo. To storimo tako, da deklariramo atribut "login-required" v deskriptorju strani pages.xml.

```
1 <page login-required="true"/>
```

V primeru, da Seamu nismo povedali, kje se nahaja prijavna stran, uporabnik dobi napako "NotLoggedInException". Seveda je smiselno to predhodno konfigurirati, kar naredimo z deklaracijo v globalnem deskriptorju strani.

```
<pages login-view-id="/login.xhtml" />
```

Nekatere aplikacije zahtevajo, da je uporabnik avtenticiran preden lahko dostopa do katere koli strani znotraj aplikacije. V tem primeru je smiselno poenostaviti konfiguracijo in se s tem izogniti deklaraciji za vsako stran posebej. V tem primeru je prijavna stran tudi prva stran aplikacije in takšna poenostavitev je navedena na primeru izvorne kode 3.12.

```
<pages login-view-id="/login.xhtml">
   <page view-id="*" login-required="true"/>
</pages>
```

Izvorna koda 3.12: Omejitve dostopa do strani

Seam avtomatsko ugotovi, katera je prijavna stran in dovoli nezaščiten dostop do nje. To je pomembno, saj v nasprotnem primeru uporabnik ne bi imel možnosti avtentikacije.[15]

Uporaba HTTPS povezave za dostop do aplikacije

Drugo področje, ki ga je potrebno omeniti, je varna komunikacija oz. kriptirano posredovanje strani. V spletnih aplikacijah za te namene uporabljamo protokol HTTPS, ki kriptira celoten promet, ki ga strežnik pošilja ali prejme z uporabo protokola SSL. Uporaba varne povezave je razvidna iz predpone URL naslova, ki mora biti HTTPS. Takšen način komunikacije konfiguriramo preko deskriptorja strani, kjer določimo, ali sprejema HTTP ali HTTPS zahteve. V Seamu bi zahtevali uporabo protokola HTTPS na prijavni strani z naslednjo deklaracijo:

```
<page view-id="/login.xhtml" scheme="https"/>
```

V primeru, da želi uporabnik dostopati do prijavne strani preko protokola HTTP, ga Seam avtomatsko preusmeri na HTTPS. Če za nadaljnje zahteve ni specificiran tip protokola, bo uporabnik dostopal do vseh naslednjih strani preko protokola HTTPS. Seam bo obdržal nastavitev iz prejšnjega zahtevka.[15]

3.6.3 Konfiguracija preverjanja vlog

Pojem avtentikacije in avtorizacije uporabniki radi zamešajo. Avtentikacija preverja identiteto uporabnika, avtorizacija pa preverja, kaj lahko uporabnik vidi in katere vire ima na voljo. Tudi ko je uporabnik avtenticiran, je potrebno preveriti njegove pravice, saj bi bili v nasprotnem primeru administratorji vsi.

V zgornjih odstavkih smo omenili, da ima uporabnik oz. objekt Identity določene vloge. Te vloge ločujejo uporabnika od tistega, ki teh vlog nima. Temu načinu pravimo avtorizacija na podlagi vlog. Tako lahko vsakemu uporabniku zelo natančno določimo, do katerih virov lahko dostopa in do katerih ne. Pri Seamu lahko takšne omejitve postavimo na stran, komponento ali metodo z uporabo anotacij. Takšno preverjanje se zanaša na Expression Language (EL), ki je specifičen programski jezik, večinoma namenjen javanskim spletnim aplikacijam. Med drugim omogoča izvajanje programske kode v spletnih straneh. Pri navadnem EL-ju to ne bi bilo mogoče, saj lahko takšna koda dostopa samo do spremenljivk na zrnih EJB, zato Seam uporablja JBoss EL, ki omogoča klic metod s parametri. Na takšen način metodi podamo kot argument ime vloge, ki jo preverja.

Tukaj gre Seam še korak naprej. V EL-ju ima registrirani še dve dodatni funkciji s:hasRole in s:hasPermission, ki omogočata preverjanje vlog kar v JSF-ju. Funkcija s:hasRole se mapira v statično metodo definirano z izvorno kodo 3.13.

```
1 public static boolean hasRole(String name) {
2   return Identity.instance().hasRole(name);
3 }
```

Izvorna koda 3.13: Seamova metoda hasRole

Funkcija s:hasPermission ima podobno implementacijo. Glede na to, da se izvajanje preverjanja prenese na komponento identity, lahko zadevo poenostavimo in izvedemo #{identity.hasRole('admin')}.

Deklaracija omejitev postavlja varnostne omejitve na raznih mestih po aplikaciji. Če hoče uporabnik mimo varnostne omejitve, mora biti avtenticiran in zadovoljiti varnostne zahteve. Če uporabniku dostop ni omočen, ga aplikacija preusmeri na prijavno stran ali vrže napako. Seam ponuja dva načina omejitev, enega za varovanje strani JSF, drugega za varovanje komponent. Oba sta sposobna preverjanja na podlagi vlog in na podlagi pravil. V tem sklopu se bomo osredotočili na varnost na podlagi preverjanja vlog.

Če uporabniku skrijemo povezavo, do katere nima pravice, preprečimo, da bi zašel na napačno pot in se spraševal, zakaj nima tega dostopa. Vendar to ni dovolj, saj uporabniki vedno najdejo načine, da obidejo takšne preproste omejitve. Da dosežemo omejitve na nivoju strani, uporabimo element <restrict> v vozlišču <page> v deskriptorju strani. Seam nato preverja te omejitve v dveh fazah JSF-ja, in sicer pred fazo "Restore View" in "Render Response". V primeru, da preverjanje spodleti, se sproži izjema "NotLoggedInException" ali "AuthorizationException", odvisno, ali je uporabnik prijavljen v aplikacijo ali ne. Napako "AuthorizationException" lahko ulovimo in nato uporabnika preusmerimo na stran, kjer mu prikažemo, da je prišlo do napake. To nam omogoča naslednji del kode.

```
<exception class="org.jboss.seam.security.
    AuthorizationException">
  <redirect view-id="/securityError.xhtml">
    <message severity="warn">You've been denied access to the
       resource.</message>
  </redirect>
</exception>
```

Izvorna koda 3.14: Deklaracija izjeme v pages.xml

Če želimo uporabnikom, ki niso administratorji, preprečiti dostop do administratorske sekcije, lahko to naredimo tako, da uporabimo element <restrict> na način, kot je naveden spodaj. Spodnja omejitev bo pri pametni razmestitvi kode delovala za celotno administratorsko sekcijo brez, da bi definirali omejitev za vsako stran posebej.

```
<page view-id="/admin/*">
  <restrict>#{s:hasRole('admin')}</restrict>
</page>
```

Izvorna koda 3.15: Omejitev na nivoju strani

Kot vidimo, vsebuje zgornji element <restrict> izraz iz EL-ja. Ta izraz mora vedno vračati boolean vrednost. Toda varovanje strani je šele polovica zgodbe. Če uporabniku kakor koli uspe zaobiti varnost na nivoju strani in mu uspe priti do komponent preko kakšne druge poti (npr. spletne storitve), moramo zagotoviti, da so tudi te ustrezno varovane. Seam nam omogoča, da deklarativno zavarujemo komponente na podoben način, kot poteka varovanje strani. Seamove komponente zavarujemo z uporabo anotacije @Restrict na nivoju metode ali razreda. Omejitev se navede z uporabo EL-ja, ki lahko sprejme funkcijo s:hasRole ali s:hasPermission ali pa preveri, če je uporabnik avtenticiran z #{identity.loggedIn}. Če preverjanje spodleti, Seam vrže isto

izjemo kot pri spodletelem preverjanju na strani.

```
@Name("adminComponent")
@Restrict("#{s:hasRole('admin')}")
public class AdminComponent {
  public void grantAccess() { ... }
  public void anotherAdminAction() { ... }
}
```

Izvorna koda 3.16: Definicija omejitev na komponenti

V zgornjem primeru vidimo, da je omejitev definirana na razredu. Vse metode, ki nimajo definirane anotacije @Restrict to omejitev podedujejo od razreda. Zraven naj še omenim, da se ta anotacija lahko uporablja tudi na entitetah, kar omogoča, da se preverjajo pravice, preden se instance persistirajo.

Omejitev @Restriction se na razredu ali metodi prevede v izraz Identity.instance().checkRestriction("#{s:hasRole('admin')}"); ali Indentity.instance().checkRole("admin");. Ta omejitev pa ne preverja samo pravic uporabniku, ampak izvede tudi določene akcije v primeru spodletelega preverjanja. Seveda se lahko razvijalec odloči, da prepusti Seamu, naj vrže izjemo, katero nato prikažemo uporabniku ali sesujemo aplikacijo, lahko pa še prej zabeležimo takšne neavtoriziran poskus prijave. Takšno beleženje nam lahko kasneje pomaga odkriti napade na našo aplikacijo.

V določenih primerih pa naletimo tudi na takšen scenarij, ko se mora metoda izvesti kljub temu, da uporabnik nima potrebnih pravic. Tukaj je ideja, da ne omejimo uporabnika, temveč scenarij, po katerem se lahko metoda izvede. Seam to podpira preko razreda RunAsOperation. Ideja je povzeta po Unixovi sudo komandi in omogoča začasen dvig privilegijev uporabniku. Koda se izvede kot sistemska operacija, kar zaobide vsa preverjanja in omejitve.

```
new RunAsOperation() {
  public String[] getRoles() {
    return String[] {"admin"};
  }
  public void execute() {
    facilityAction.setOwner(user);
  }
}
```

Izvorna koda 3.17: Uporaba RunAsOperation začasno dvigne uporabnikove privilegije

Zgornji primeri nam kažejo, kako preprosta je avtorizacija na podlagi pravil. Uporabna je za varovanje določenih sekcij aplikacije, kot je na primer administracijska sekcija, vendar bomo za varovanje večuporabniške aplikacije načeloma potrebovali varnost, ki bazira na pravilih iz konteksta.[15]

3.6.4 Zaščita aplikacije pred roboti

Večino časa se ukvarjamo z varnostjo proti vsiljivcem, toda moramo se zavedati, da se lahko zlorabijo tudi javno dostopni viri. Zlonamerni programi lahko zlorabijo registracijsko formo, vnašajo naključne uporabnike, polnijo podatkovno bazo z lažnimi podatki in vodijo do tega, da se onemogoči dostop pravim uporabnikom. V te namene se uporablja CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Appart). Ta skuša razpoznati, ali je odjemalec človek ali računalnik. To lahko implementiramo na različne načine, splošna ideja pa je, da aplikacija postavi izziv, ki ga mora uporabnik rešiti tipično tako, da uporabniku izriše sliko. Samo v primeru, da uporabnik reši izziv pravilno, se forma pošlje nazaj na strežnik. Izziv za odjemalca zahteva osnovne človeške sposobnosti, ki jih še najbolj sofisticiran program ne more imeti, zato se v primeru uspešno rešenega izziva predpostavi, da ga je rešil človek.

Seam implementira CAPTCHA tako, da izriše osnovni aritmetični problem z uporabo Jave 2D. S celotno interakcijo upravlja Seam in vse, kar mora programer storiti, je, da to doda v formo in v aplikacijo namesti SeamResourceServlet, ki skrbi za izgradnjo slike. Preverjanje omogočimo tako, da povežemo vnosno polje z vrednostjo #{captcha.response}, v katero odjemalec vnese odgovor.

```
1 <s:decorate id="verifyCaptcha" template="layout/edit.xhtml">
2   <ui:define name="label">Security check</ui:define>
3     <h:graphicImage value="/seam/resource/captcha"/>
4   <h:inputText value="#{captcha.response}" required="true"/>
5 </s:decorate>
```

Izvorna koda 3.18: Integracija Seam CAPTCHA

Kot vidimo iz zgornjega primera izvorne kode 3.18, se uporablja Seamova komponenta CAPTCHA, ki skrbi za delovanje preverjanja. Validacija deluje na podlagi Hibernateovega validatorja za anotacijo @CapchaRespose, ki je definiran na polju response. Za uporabo privzete Seamove implementacije preverjanja CAPTCHA je to celotna nastavitev, ki jo potrebujemo, opcijsko pa lahko programer aplikacije razširi razred org.jboss.seam.captch.Captcha in implementira lastno preverjanje.

Borba med varnostnimi mehanizmi in zlonamernimi programi je stalno v teku in vedno moramo skrbeti za nadgrajevanje naših varnostnih tehnologij, saj se tudi zlonamerni programi nenehno izpopolnjujejo, postajajo vedno zmogljivejši in odkrivajo nove varnostne luknje v naših mehanizmih za varovanje aplikacij.[15]

3.6.5 Povzetek

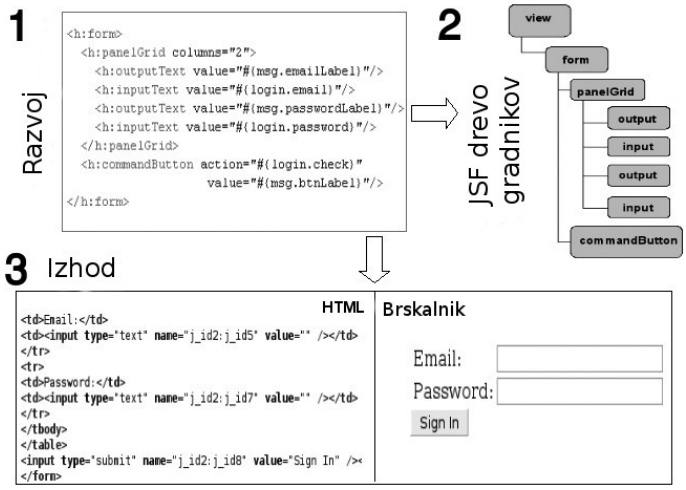
Implementacija varnostnih mehanizmov je v Seamu preprosta. Samo ena varnostna pregrada ni dovolj, saj potrebujemo varnost na vseh nivojih, na

straneh, na komponentah in na entitetah, kar nam omogoča, da so, če uporabnik uspe zaobiti prvo varnostno linijo, zadaj druge, ki čakajo nanj. Videli smo, da lahko avtentikacijo dodamo praktično samo z implementacijo ene metode v objektih POJO, ki validira podatke, kateri pridejo iz komponente Identity. Seam in JAAS delata skupaj in skrbita za zagotavljanje nizkonivojske varnosti. Poleg tega komponenta Identity sprejema vloge, preko katerih lahko izvajamo preverjanje avtorizacije po celotni aplikaciji in preprečujemo uporabniku dostop do virov, do katerih ni upravičen. Nazadnje vidimo, kako preprosta je implementacija preverjanja CAPTCHA, ki je prav tako pomembno in ščiti naše javno izpostavljene vire.

3.7 JavaServer Faces

JavaServer Faces ali na kratko JSF je spletno ogrodje, ki definira tri plasti: arhitekturo komponent, nabor gradnikov uporabniškega vmesnika in aplikacijsko infrastrukturo. Omogoča uporabo integriranih standardnih gradnikov uporabniškega vmesnika, kot so gumbi, povezave, vnosna polja, izbirna polja ipd., hkrati pa omogoča integracijo teh gradnikov v kompleksnejše gradnike, ki se uporabljajo v drugih ogrodjih, kot je npr. Rich Faces, ali pa uporabnik izdela lastne gradnike.

JSF zaradi svoje narave spletno usmerjene tehnologije omogoča podporo za različne odjemalce, kot so različni brskalniki, telefoni, tablice. Arhitektura ogrodja omogoča prikaz gradnikov na različne načine tako, da ustrezajo odjemalcu. Omogoča tudi validacijo uporabniških vnosov in pretvorbo vnosov v različne objekte. Dodatno skrbi za sinhronizacijo komponent uporabniškega vmesnika z javanskimi zrni, na katere so komponente povezane, se odziva na dogodke, ki jih z interakcijo sproži uporabnik, omogoča sistem navigacije med stranmi in ima polno podporo za internacionalizacijo. Vse te funkcionalnosti omogočajo osnovno platformo, na kateri lahko razvijalci začnejo graditi spletno aplikacijo.



Slika 3.9: Pretvorba JSF kode v HTML [19]

Ključna razlika v primerjavi z ogrodji za gradnjo uporabniškega vmesnika za namizne odjemalce je, da JSF deluje na spletnem strežniku (npr. Apache Tomcat, ki je osnova aplikacijskemu strežniku JBoss) in generira datoteko HTML, ki jo prikaže uporabnikov odjemalec. [18]

3.7.1 Življenjski cikel ogrodja JSF

Ključnega pomena za izgradnjo kvalitetne in robustne aplikacije z uporabo JSF-ja je poznavanje faz življenjskega cikla zahteve in zavedanje razvijalca, kaj se v vsakem izmed teh korakov dogaja. Cikel se začne takoj, ko JSF Servlet prejme zahtevek, nato pa sledi šest faz:[18]

1. Faza "Restore View" je prva faza, ki se izvede. Glavna naloga te faze je najti pogled, ki ga uporabnik pregleduje, in prenesti uporabnikov vnos nanj. Ta pogled se lahko nahaja v uporabnikovi seji na strežniku ali v uporabnikovem odjemalcu.

2. Druga faza "Apply Request Values" izvede dekodiranje. Poslane vrednosti se nastavijo na komponente, ki so povezane na vnosna polja. Če gradnik grafičnega vmesnika definira uporabo pretvornika, se ta izvede

pred apliciranjem vrednosti na komponente, vključno s preverjanjem vrednosti v pretvorniku in propagacijo napak za neveljavne vrednosti. V tej fazi se beležijo dogodki, ki jih sprožijo povezave. Tej dogodki se dodajo v FacesContext za kasnejše procesiranje v fazi "Invoke Application". Poleg tega se vsi dogodki pošljejo ustreznim poslušalcem, ki lahko povzročijo prehod cikla na zadnjo fazo "Render Response".

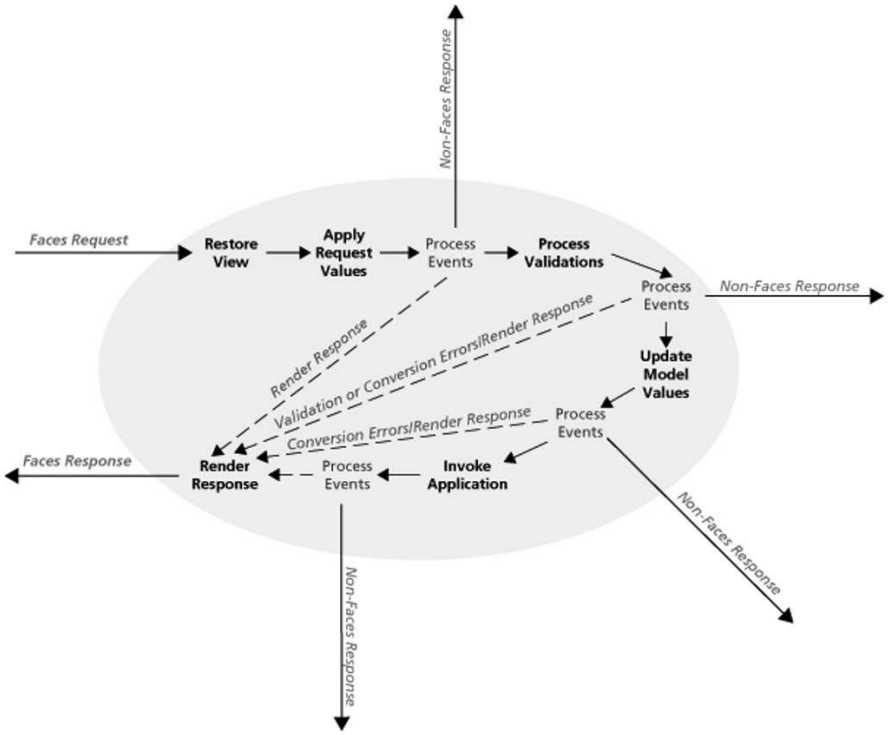
3. Tretja faza "Process Validators" izvaja, kot pove že ime samo, validacijo. To pomeni, da komponenta validira vhod najprej z lastnim validatorjem, predpisanim za to komponento, nato preveri, ali je vrednost zahtevana, nazadnje pa izvede še validacijo z zunanjimi validatorji, če so tej navedeni. V primeru uspeha se nastavi lokalna validirana vrednost, v nasprotnem primeru se zgodi prehod v zadnjo fazo "Render Response", skupaj z generirano validacijsko izjemo.

4. V četrti fazi "Update Model Values" se izvede nastavljanje validiranih vrednosti na javanska zrna, ki so povezana z gradniki. Vsak gradnik ima nastavljeno, na katero zrno se njegova vrednost preslika, in ko smo prepričani, da je vrednost ustrezna in pravega tipa, se ta lahko propagira na zrno, do katerega ima aplikacija dostop. Pomembno je, da je vse akcije, ki so bile izvedene do zaključka te faze, izvedlo ogrodje JSF avtomatsko.

5. V peti fazi "Invoke Application" se začne izvajat aplikacijska koda. Dogodki ustvarjeni v "Apply Request Values" se razpošljejo na ustrezne poslušalce in začne se izvajati metoda, ki je navedena v gradniku, ki je povzročil pošiljanje forme in izvedbe celotnega cikla JSF. V večini primerov ta metoda izvede nekakšno procesiranje, ki vrne rezultat, s katerim sistem navigacije povzroči prehod na drug pogled.

6. Zadnja faza "Render Response" poskrbi za izris izbranega pogleda. Celoten postopek apliciranja vrednosti, validacije in procesiranja je bil izveden in glavna naloga te faze je, da uporabniku vrne odgovor. Spletni strežnik pošlje uporabniku podatke, ki jih njegov brskalnik prikaže

v obliki spletne strani. Poleg tega ta faza shrani trenutno stanje, ki se bo potrebovalo v prvi fazi "Restore View" ob naslednji sprožitvi JSF cikla.



Slika 3.10: JSF-jev življenjski cikel [19]

3.7.2 Validacija vnosnih polj

Validacija uporabniškega vnosa je še ena pomembna lastnost spletnih aplikacij. Ogrodje JSF ima integrirano podporo za validacijo vnosa. Komponente, ki sprejemajo neveljavne vhode in vračajo nesmiselne izhode, vsekakor škodujejo ugledu aplikacije. JSF nima podpore za validacijo na strani odjemalca in ne generira kode JavaScript, vsa validacija se izvaja na strežniku v javanskih zrnih, standardnih validatorjih JSF in validatorjih, ki jih definira razvijalec. V primeru, da validacija zavrne vneseno vrednost, odjemalec dobi sporočilo

v obliki HTMLMessage, tretja faza JSF cikla se prekine in objekti, povezani z gradniki, se ne posodobijo.

Ločino dva tipa validacije. Validacijske metode, definirane na gradniku, so ponavadi specifične za aplikacijo in namensko narejene samo za določene primere. Po drugi strani imamo vgrajene validatorje, ki so generični in primerni za splošno uporabo. Specifične validatorje definiramo na gradniku z atributom "validator", kjer z uporabo Expression Language povemo, katera metoda naj se izvede, medtem ko standardne validatorje uporabljamo z gnezdeno značko <f:validator>, kjer za atribut validatorId navedemo ime validatorja, na katerega se sklicujemo. Imamo pa tudi možnost, da metodo, ki jo uporabljamo za validacijo vhoda, spremenimo v standardni validator. To naredimo tako, da razširimo razred javax.faces.validator.Validator. Validatorju določimo ime, na katerega se bomo sklicevali, in implementiramo metodo validate, v kateri se izvrši preverjanje.[18]

3.7.3 Avtentikacija in avtorizacija

Naslednji pomembni stvari pri spletni aplikaciji sta avtentikacija in avtorizacija uporabnika. JSF omogoča implementacijo avtentikacije s strani razvijalca, ki deluje na podoben princip kot validatorji. Metoda preverja uporabniško ime in geslo uporabnika, če le-to ne ustreza, pa ustvari izjemo, ki izpiše uporabniku obvestilo in prepreči izvedbo navigacijskih pravil. Ponuja pa tudi možnost avtorizacije, osnovane na vsebniku EJB. V sklopu te diplomske naloge avtentikacija JSF-ja ni pomembna, ker za vse te korake poskrbi ogrodje Seam, ki to rešuje na precej enostavnejši, robustnejši in predvsem preverjen način.[18]

3.8 RichFaces

RichFaces je odprto kodno ogrodje, ki doda funkcionalnost Ajaxa obstoječim aplikacijam JSF. Dopolnjuje funkcije ogrodja JSF, vključno z njegovim življenjskim ciklom, validacijo, pretvorbo vrednosti in uporabljanjem statičnih

in dinamičnih virov. Komponente ogrodja RichFaces imajo vgrajeno podporo za Ajax in omogočajo enostavno zamenjavo izgleda, predvsem pa omogočajo enostavno vgradnjo v aplikacijo JSF. Lahko bi rekli, da je RichFaces razširitev funkcionalnosti ogrodja JSF.

RichFaces nima vgrajenih lastnih varnostnih mehanizmov in nima nikakršnega prispevka k varnosti aplikacije, saj bazira na ogrodju JSF. Tako je validacija vnosa uporabnika v domeni ogrodja JSF, prijava v aplikacijo pa prav tako (v sklopu te diplomske naloge te naloge izvaja Seam). Kljub temu pa je vredno nameniti nekaj besed tudi ogrodju RichFaces, saj je ključni element pri izgradnji uporabniškega vmesnika. S svojimi gradniki, obogatenimi z Ajax podporo, omogoča razvijalcu enostavno izdelavo interaktivnega grafičnega uporabniškega vmesnika.[20]

3.8.1 JSF, Ajax, RichFaces

Izkaže se, da JSF in Ajax skupaj zelo dobro funkcionirata. JSF zagotavlja model za razvoj grafičnega vmesnika spletnih aplikacij. Razvijalcu tako ni več potrebno skrbeti za HTML kodo, ampak se lahko usmeri na izgradnjo grafičnega vmesnika iz gradnikov JSF. JSF omogoča podoben način razvoja na spletu, kot ga omogoča Swing pri namiznih aplikacijah.

Ajax, kar pomeni Asynchronous JavaScript and XML, v bistvu ni ogrodje, ampak tehnika za izgradnjo interaktivnih spletnih aplikacij. Ajax je sestavljen iz DHTML, XML, XMLHTTPRequest in Document Object Model (DOM), kar skupaj omogoča izgradnjo bogatejših aplikacij, ki delujejo znotraj brskalnika. Dodatna prednost Ajaxa je v tem, da uporabnik ne potrebuje nobenega dodatnega vtičnika v svojem brskalniku.

Spletne aplikacije, ki bazirajo na Ajaxu so interaktivnejše in hitrejše. Vedno bolj se približujejo namiznim aplikacijam v smislu možnosti, ki so ponujene uporabniku. Glavna ideja oz. tehnika za tem je, da se posodobi samo tisti del strani, ki je spremenjen. To pomeni, da ni več potrebe po osvežitvi celotne strani, posledično pa imamo hitrejše delovanje in več možnosti za interakcijo.

Razlog, zakaj JSF in Ajax tako dobro funkcionirata skupaj, je v modelu komponent JSF. Tukaj nastopi RichFaces, ki je, kot že ime pove, bogata knjižnica komponent JSF, ki ponuja komponente z vključeno Ajax podporo. Tako imamo komponente, ki zajemajo ves potreben JavaScript in ostale tehnologije, potrebne za delovanje Ajaxa. Razvijalcu se tako ni treba ukvarjati z ročnim razvojem Ajaxa, ker je vsa ta logika že razvita in vgrajena v samih komponentah. Dodatna prednost je v tem, da so vse te komponente preverjene za delovanje na vseh brskalnikih.[19]

3.8.2 Ajax4jsf in RichFaces

Ajax4jsf ima svoje korenine v knjižnici RichFaces. Ajax je bil ob nastajanju RichFaces nova popularna tehnologija in cilj projekta Ajax4jsf je bil enostavna uporaba z obstoječimi komponentami JSF. Tako se ogrodje uporablja za ponovno procesiranje, kjer se samo določen del DOM-a posodobi in ponovno izriše želeni del strani.[19]

Poglavje 4

Opis rešitve

4.1 Izbira tehnologij

Ena izmed naročnikovih zahtev je bila integracija spletne aplikacije Dnevniki odjema odjemalca z obstoječo aplikacijo Obračun prenosa zemeljskega plina (OPZP), ki v internem omrežju že deluje. Aplikacija OPZP bazira na informacijskem sistemu GemaLogic podjetja Solvera Lynx d. d., kar pomeni, da je bila izbira tehnologij določena že vnaprej. Tako smo pri razvoju spletne aplikacije Dnevniki odjema odjemalca uporabili aplikacijski strežnik JBoss 4.2.2.GA in njegovo knjižnico za spletne storitve JBossWS. Za shranjevanje podatkov se uporablja Microsoft SQL Server 2008 R2 in Hibernate 3.3.1.GA kot ORM, ki upravlja s persistiranimi javanskimi objekti. Aplikacijski strežnik JBoss za pridobivanje podatkov in shranjevanje novih uporablja ogrodje EJB 3.0, ki v kombinaciji z Hibernate-om, kot implementacijo JPA standarda, skrbi za pretok podatkov med strežnikom in bazo. Za izdelavo grafičnega vmesnika je uporabljeno ogrodje JSF 1.2.04 in na njem temelječa knjižnica RichFaces 3.3.3.Final. Za povezavo in pretok podatkov med grafičnim vmesnikom in zaledjem, ki pridobiva in obdeluje podatke, skrbi Seam 2.2.2.Final, ki v rešitvi nastopa kot ključni dejavnik, ki zagotavlja integracijo med različnimi ogrodji. Dodatno razvijalcu ponuja še varnostne mehanizme, ki skrbijo za varnost na vseh nivojih spletne aplikacije. Aplikacijski strežnik

JBoss omogoča izvajanje opravil po urniku, ki v spletni aplikaciji Dnevniki odjema odjemalca skrbijo za uporabnika neviden pretok podatkov proti spletni aplikaciji OPZP. To omogoča knjižnica Quartz 1.7.3, ki na aplikacijskem strežniku izvaja opravila po urniku. Vse skupaj se izvaja na javanskem virtualnem stroju verzije 1.6.

4.2 Izdelava spletne aplikacije

Izdelava spletne aplikacije je potekala v več korakih, saj je bilo potrebno razviti različne nove komponente, ki skrbijo za pridobivanje podatkov o dnevnikih odjema, omogočajo uporabniku prikaz, vnos in urejanje novih dnevnikov odjema in prenos uporabniško vnesenih vrednosti v sistem OPZP. V sklopu izdelave rešitve smo razvili nove spletne storitve, nov grafični vmesnik in novo opravilo za prenos podatkov.

4.2.1 Prijava v aplikacijo

Prijavo v aplikacijo upravlja ogrodje Seam in ni bila razvita v sklopu projekta Dnevniki odjema odjemalca, saj je del informacijskega sistema GemaLogic. Glede na to, da je prijava v aplikacijo pomemben element pri zagotavljanju varnosti, mu bomo vseeno namenili nekaj besed.

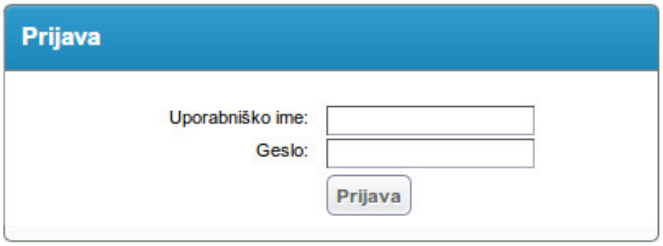
Seamova prijava bazira na metodi authenticate(). Dostop do metode je definiran v deskriptorju components.xml, tako kot ogrodje Seam to predpisuje. Metoda se lahko nahaja v poljubnem razredu, edini zahtevi sta dostopni modifikator tipa public in vračanje vrednosti boolean.

V prijavni formi na sliki 4.1, izdelani z ogrodjem JSF, uporabnik vnese uporabniško ime in geslo v vnosni polji, podatki pa se prepišejo na Seamovo komponento Identity. Ob predložitvi forme se izvede metoda login() na komponenti Identity. V postopku prijave se izvede metoda authenticate(), ki vrača boolean vrednost in pove, ali se je uporabnik uspešno avtenticiral. S pridobljenimi podatki se prebere, ali takšen uporabnik obstaja v bazi, in v primeru, da je uporabnik vnesel pravilno uporabniško ime in geslo, se vse

```
1  public boolean authenticate() {
2    User user = findUser();
3    if (user != null) {
4      for (UserGroup userGroup : user.getUserGroups()) {
5        identity.addRole(userGroup.getGroup().getName());
6      }
7      return true;
8    }
9    log.info("Login unsuccessful: #0", identity.getUsername());
10   return false;
11 }
```

Izvorna koda 4.1: Avtentikacijska metoda

njegove vloge prepišejo na objekt Identity. Vse to izvaja izvorna koda 4.1. Na takšen način lahko nato Seam avtomatsko izvaja avtorizacijo, kjer je definirana anotacija @Restrict.



Slika 4.1: Prijava v aplikacijo Dnevniki odjema odjemalca

V izvorni kodi 4.2 lahko vidimo, kako je izdelana prijava forma.

```
<table cellpadding="1" cellspacing="0" class="labelValueUI">
  <tr><td>
      <h:outputLabel for="password" value="#{messages.username}
          "/>:
    </td>
    <td>
      <h:inputText id="username" value="#{identity.username}"
          onkeypress="submitForm(event);"/>
    </td>
  </tr>
  <tr><td>
      <h:outputLabel for="password" value="#{messages.password}
          "/>:
    </td>
    <td>
      <h:inputSecret id="password" value="#{identity.password}"
           onkeypress="submitForm(event);"/>
    </td>
  </tr>
  <tr>
    <td></td>
    <td>
      <h:commandLink accesskey="enter" id="submitLink"
          styleClass="button" action="#{identity.login}">
        <span>#{messages['command.login']}</span>
      </h:commandLink>
    </td>
  </tr>
</table>
```

Izvorna koda 4.2: Prijavna forma

Aplikacija posluša na vtičnici HTTPS strežnika JBoss in zahteva avtentikacijo odjemalca, kar pomeni, da potrebuje uporabnik za dostop do aplikacije digitalno potrdilo, ki ustreza javnemu ključu, ki se nahaja v varni shrambi aplikacijskega strežnika JBoss. V primeru, da je uporabnikovo digitalno potrdilo ustrezno, razreševalnik digitalnih potrdil pridobi uporabniško ime in ga nastavi kot atribut username Seamove komponente Identity. Kljub uporabi digitalnih potrdil, je na željo naročnika uporabnik primoran vnesti geslo.

4.2.2 Razvoj spletnih storitev

Za potrebe pridobivanja dnevnikov odjema iz spletne aplikacije OPZP je bilo potrebno najprej razviti spletno storitev, ki bo aplikaciji DOO servirala podatke iz strežnika OPZP. Druga spletna storitev, ki je bila razvita, pa skrbi za prenos novo vnesenih dnevnikov odjema iz aplikacije DOO v aplikacijo OPZP. Obe spletni storitvi temeljita na zrnu EJB in tehnologijah JBossWS, ki javansko zrno izpostavijo kot spletno storitev tipa SOAP. Omenjene tehnologije ponujajo deklarativen način zagotavljanja varnosti z uporabo anotacij.

Anotacija @Stateless pove strežniku, naj naš javanski objekt upravlja kot zrno EJB, medtem ko s pomočjo anotacije @WebService iz tega zrna generira WSDL in ga naredi dostopnega preko protokola HTTP. Anotacija @WebContext upravlja z varnostjo spletnih storitev. Glede na zahteve naročnika o zagotavljanju varnosti smo definirali parametre authMethod = ”CLIENT-CERT” in transportGuarantee = ”CONFIDENTIAL”. To pomeni, da spletna storitev za avtentikacijo zahteva uporabo digitalnega potrdila in vsa komunikacija se tako šifrira in poteka preko protokola HTTPS.

Ko je odjemalec spletnih storitev uspešno prijavljen v vsebnik EJB, preko mehanizmov spletne storitve zahteva dostop do ene izmed metod anotiranih z @WebMethod. Še preden pa odjemalcu dovolimo dostop do takšne metode, moramo izvesti postopek avtorizacije. Postopek avtorizacije se izvede s pomočjo anotacije ogrodja EJB @RolesAllowed. To pove ogrodju, da mora odjemalec imeti eno izmed pravic, definiranih na anotaciji. Anotacija se lahko uporabi na celotnem razredu, kar pomeni, da velja za vse metode razreda, ali

pa na vsaki metodi posamično. Tako pri pridobivanju podatkov iz OPZP-ja, kot pri branju uporabniško vnesenih dnevnikov v spletni aplikaciji DOO, smo definirali anotacijo @RolesAllowed na nivoju razreda, saj en proces dostopa do vseh metod javanskega zrna. Postopki, opisani v zgornjih dveh odstavkih, se izvajajo na podlagi izvorne kode 4.3.

```
1  @Stateless
2  @WebService(name = "GBCustomerGasDiary",
3              targetNamespace =
4      "http://www.solveralynx.com/gema/webservice/customergasdiary",
5              serviceName = "CustomerGasDiaryService")
6  @WebContext(authMethod = "CLIENT-CERT",
7              transportGuarantee = "CONFIDENTIAL",
8              secureWSDLAccess = false,
9              contextRoot = "gemaws",
10             urlPattern = "/customergasdiary")
11 @RolesAllowed(value = {"CGDWebServiceUser"})
12 public class GBCustomerGasDiaryEJB implements GBCustomerGasDiary {
13   ...
14 }
```

Izvorna koda 4.3: Definicija spletne storitve za pridobivanje podatkov iz aplikacije OPZP

Naslednje preverjanje, ki se izvaja pri dostopu do podatkov v aplikaciji OPZP, je vsebinsko. Preko sistema pogodb o dostopu, bilančnih pogodb in pogodb o prenosu se preveri, ali je neka pravna oseba, ki jo predstavlja odjemalec, upravičena do podatkov merilnega mesta, ki jih želi. Vse pogodbe v aplikaciji OPZP imajo seznam merilnih mest, ki jih zajemajo, in veljavnost pogodbe. Tako logika, ki je bila specifično razvita za zagotavljanje, da lahko vsaka pravna oseba pridobi samo podatke, do katerih je upravičena, filtrira zahtevke, do katerih odjemalec ni upravičen. Na tak način smo zagotovili, da ne more naključni uporabnik dostopati do informacij v lasti neke druge

pravne osebe. Spletna storitev vrne tudi trajanje pogodb v izbranem obdobju. Ta informacija je uporabljena v grafičnem vmesniku aplikacije DOO, kjer uporabniku omogoča oz. onemogoča vnos podatkov za izbrano časovno obdobje.

Spletna storitev na strani DOO takšnega vsebinskega preverjanja ne vsebuje, saj do nje dostopa zgolj opravilo iz aplikacije OPZP, ki prenaša uporabniško vnesene dnevnike iz sistema DOO v OPZP. Ta spletna storitev opravlja zgolj vlogo transporta med aplikacijama in beleži, kateri dnevniki so že bili poslani, da se prepreči podvojeno pošiljanje. Pošiljajo se samo dnevniki, ki so s strani uporabnika potrjeni za pošiljanje. Na tak način preprečujemo možnost, da bi se prenesli podatki, katerih uporabnik ni želel še poslati.

4.2.3 Razvoj grafičnega vmesnika

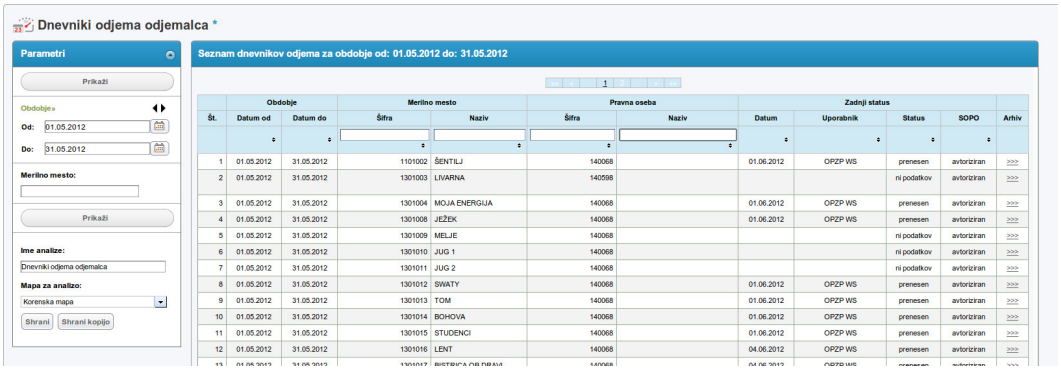
Grafični vmesnik aplikacije DOO smo razvili z uporabo spletnega ogrodja JSF, knjižnice RichFaces, ki predstavlja grafične elemente na osnovi ogrodja JSF, Ajaxa, ogrodja Seam in JBossove knjižnice JAX-WS, ki smo jo uporabili za izdelavo odjemalca za dostop do spletne storitve za pridobivanje podatkov iz aplikacije OPZP.

Prvi varnostni mehanizem, ki je opisan z izvorno kodo 4.4, ki omejuje dostop do orodja, je Seamova avtorizacija. Kot je opisano zgoraj, Seam deklarativno izvaja avtorizacijo in od uporabnika zahteva, da ima vlogo "customerGasDiary", saj v nasprotnem primeru Seam uporabniku prepreči navigacijo do orodja za pregled in urejanje dnevnikov odjema.

Spletna aplikacija uporabniku ponudi seznam vseh njegovih merilnih mest, kot je prikazano na sliki 4.2, za katere lahko uporabnik vnaša odjeme v izbranem obdobju. Kombinacija merilnega mesta in izbranega meseca predstavlja dnevnik odjemov za mesec. Izpišejo se vsi že obstoječi dnevniki in tisti, ki jih uporabnik v izbranem obdobju mora še vnesti. Ob izboru vrstice se uporabniku prikaže tabela z dnevi v tem mesecu in vnosnimi polji, v katere uporabnik vnaša odjeme. V primeru, da dnevnik še ne obstaja, se generira nov.

```
1 @Name(value = "customerGasDiaryReportType")
2 @Scope(value = ScopeType.CONVERSATION)
3 @Restrict(value = "customerGasDiary")
4 public class CustomerGasDiaryReportType implements ReportType<
      CustomerGasDiaryReportParams> {
5 ...
6 }
```

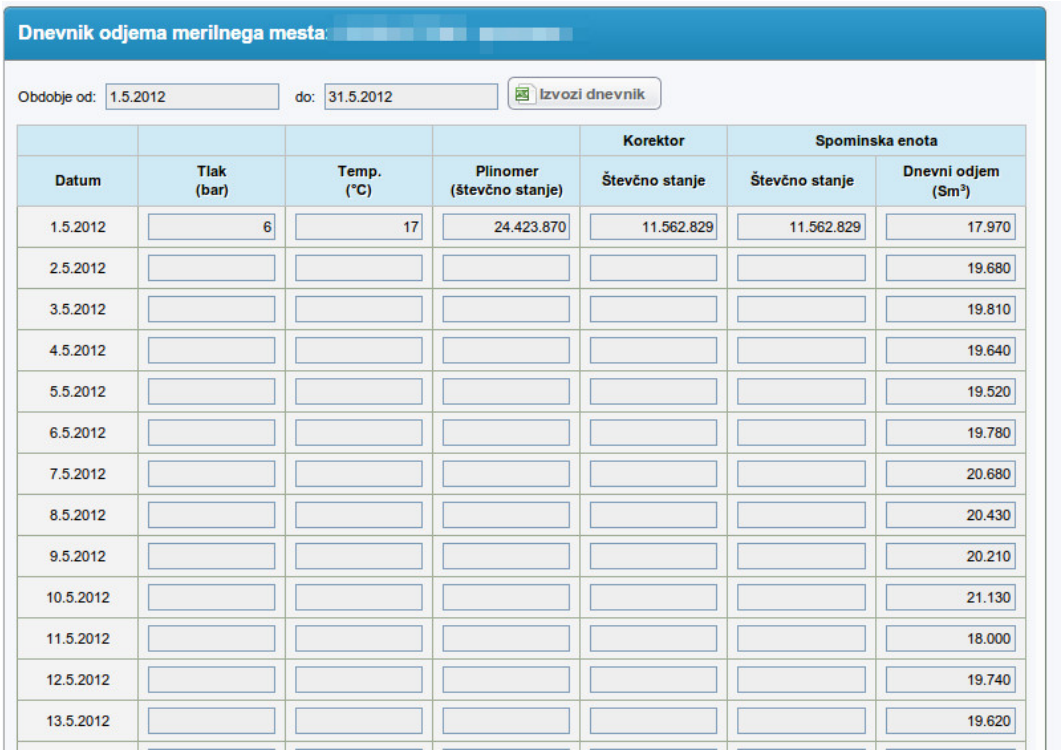
Izvorna koda 4.4: Seamova komponenta, ki pridobiva podatke za grafični vmesnik



Slika 4.2: Seznam dnevnikov odjema, do katerih lahko dostopa prijavljen uporabnik

Uporabnik ima možnost pregledovanja vseh svojih dnevnikov, ureja pa lahko le tiste, ki še ne obstajajo in hkrati niso bili ustvarjeni v aplikaciji OPZP ali niso bili še potrjeni za prenos v aplikacijo OPZP. Hkrati se uporabniku omogoči vnos zgolj na vnosnih poljih, ki pripadajo dnevom, za katere ima veljavno pogodbo v aplikaciji OPZP. Informacija o pogodbah, ki se pošlje pri pridobivanju obstoječih dnevnikov iz OPZP-ja, se tukaj uporabi za ugotavljanje, ali je uporabnik upravičen do vnosa vrednosti za določen dan ali ne. Na vseh vnosnih poljih se izvaja validacija, ki preverja, ali je uporabnik vnesel veljavno vrednost. V primeru vnosa neveljavne vrednosti se prekine

shranjevanje dnevnika in se uporabnika obvesti o neveljavnem vnosu. Kako izgledata pregled in vnos vrednosti, je prikazano na sliki 4.3.



				Korektor	Spominska enota	
Datum	Tlak (bar)	Temp. (°C)	Plinomer (števčno stanje)	Števčno stanje	Števčno stanje	Dnevni odjem (Sm^3)
1.5.2012	6	17	24.423.870	11.562.829	11.562.829	17.970
2.5.2012						19.680
3.5.2012						19.810
4.5.2012						19.640
5.5.2012						19.520
6.5.2012						19.780
7.5.2012						20.680
8.5.2012						20.430
9.5.2012						20.210
10.5.2012						21.130
11.5.2012						18.000
12.5.2012						19.740
13.5.2012						19.620

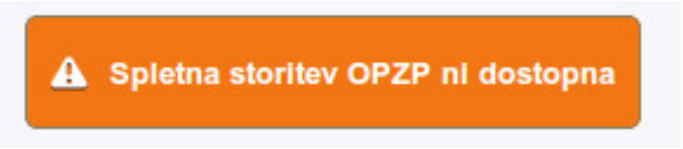
Slika 4.3: Seznam dnevov v izbranem mesecu, za katerega uporabnik vnaša podatke o odjemih.

Urejanje dnevnika je mogoče, dokler ga opravilo ne prenese v aplikacijo OPZP, nato je urejanje onemogočeno. Ko je uporabnik zadovoljen s svojim vnosom, dnevnik označi za pošiljanje, kar pomeni, da čaka na opravilo, ki ga bo preneslo v aplikacijo OPZP. V času, dokler dnevnik odjemov čaka na opravilo, ima uporabnik še vedno možnost urejanja in s tem preklic pošiljanja.

Grafični vmesnik ponuja tudi drugi način delovanja, in sicer način brez povezave. To pomeni, da strežnik OPZP ni dostopen spletni aplikaciji DOO in se uporabniku prikažejo samo lokalni dnevniki, ki so bili preneseni dokler je povezava delovala. Uporabnik ima še vedno iste možnosti pregledovanja in urejanja, kot jih ima v normalnem načinu s povezavo, le da se dnevniki, ozna-

čeni za pošiljanje, ne prenesejo, dokler se povezava ne vzpostavi. Uporabnik je obveščen z opozorilom, kot je prikazano na sliki 4.4

Ena izmed zahtev je bila pri razvoju tudi ta, da uporabnik ne sme čutiti posledic prekinjene povezave s strežnikom OPZP in mora imeti možnost opravljat svoje delo nemoteno. V primeru brez povezave ima uporabnik možnost urejanja, vendar lahko pride do scenarijev, ko bi pri ponovni vzpostavitvi povezave prišlo do nekonsistentnega stanja. Primer je, da se v tem obdobju ustvari isti dnevnik v obeh aplikacijah. Za ta primer ima opravilo, ki prenaša dnevnike, vgrajene varnostne mehanizme, ki skrbijo za razreševanje konfliktov, ki bi pripeljali do nekonsistentnega stanja.



Slika 4.4: Obvestilo o izpadu povezave do strežnika OPZP

4.2.4 Razvoj opravila za prenos dnevnikov odjema v aplikacijo OPZP

Opravilo za prenos dnevnikov, ki je definirano z izvorno kodo 4.5, predstavlja zrno EJB in je integrirano v sistem opravil informacijskega sistema GemaLogic, s katerim upravlja knjižnica za razporejanje opravil Quartz. Metoda execute(CustomerGasDiaryFetchJob job) definira vstopno točko opravila in začne njegovo izvajanje.

```
1 @Stateless
2 @Local
3 public class GBCustomerGasDiaryFetchJobEJB implements GBJob<
      CustomerGasDiaryFetchJob> {
4     ...
5
6     @Interceptors(JobInterceptor.class)
7     public void execute(CustomerGasDiaryFetchJob job) throws
          JobException {
8       ...
9     }
```

Izvorna koda 4.5: Opravilo, ki prenaša dnevnike v sistem OPZP.

Opravilo se izvaja po vnaprej določenem urniku, ki se ga določa s pomočjo orodja GemaAdministrator. Izvajanje omogoča knjižnica Quartz, ki zažene zrno EJB, ki se izvaja v vsebniku EJB. Sestavljeno je iz odjemalca spletnih storitev, generiranega z uporabo knjižnice JBoss JAX-WS, in logike, ki zapisuje podatke v bazo podatkov aplikacije OPZP.

Najprej odjemalec spletnih storitev, prikazan v izvorni kodi 4.6, dostopa do spletne storitve aplikacije DOO, iz katere prenese vse dnevnike odjemov, ki so označeni za pošiljanje. Za vsak prenesen dnevnik preveri, ali je merilno mesto, kateremu pripada, veljavno in ali je na njem omogočeno vpisovanje dnevnikov preko aplikacije DOO. Nato se preveri, ali dnevnik v bazi podatkov

OPZP že obstaja, v primeru, da ne obstaja, pa se generira nov dnevnik odjemov.

```
@WebServiceClient(
  name = "CustomerGasDiaryClientService",
  targetNamespace = "http://solveralynx.com/gema/webservice/
      customergasdiaryclient",
  wsdlLocation = "http://localhost:8081/gemaws/customergasdiaryclient
      ?wsdl")
public class CustomerGasDiaryClientServic extends Service {
  ...
  @WebEndpoint(name = "GBCustomerGasDiaryClientPort")
  public GBCustomerGasDiaryClient getGBCustomerGasDiaryClientPort() {
    return (GBCustomerGasDiaryClient)super.getPort(new QName(
      "http://solveralynx.com/gema/webservice/customergasdiaryclient"
          ,
      "GBCustomerGasDiaryClientPort"),
      GBCustomerGasDiaryClient.class);
  }
}
```

Izvorna koda 4.6: Odjemalec spletnih storitev

Naslednji korak je zapis prenesenih podatkov na dnevnike odjema v aplikaciji OPZP. V primeru, da vir podatkov na dnevniku ne ustreza prenesenim podatkom, se dnevnik zavrne in podatki niso zapisani. Če podatki ustrezajo, se preveri, ali so res prišli vsi podatki za cel mesec. Obstaja možnost, da si več različnih pravnih oseb deli isto merilno mesto v različnih obdobjih meseca. V tem primeru se podatki različnih pravnih oseb združijo v en dnevnik odjema v aplikaciji OPZP. Šele ko je dnevnik odjemov popoln, se označi kot prejet.

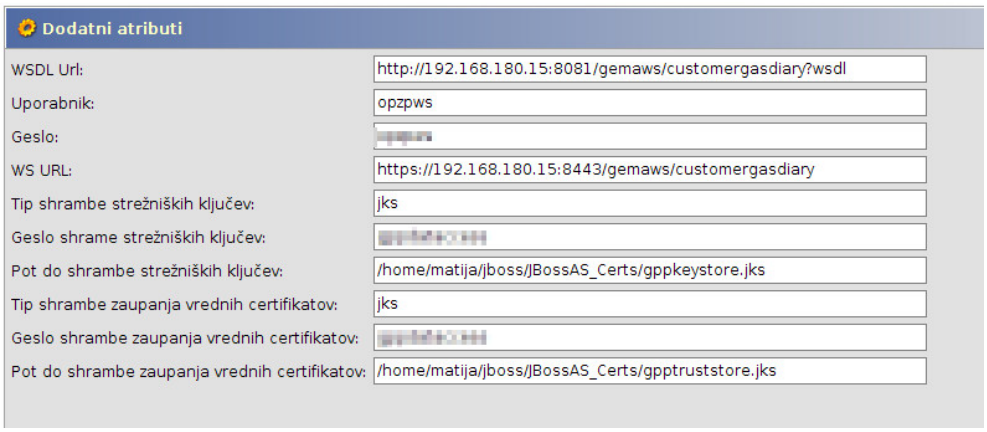
V primeru napake pri prenosu podatkov dnevnik dobi status zavrnjen, kar pove uporabniku aplikacije DOO, da je prišlo do napake. V tem primeru

se mora le-ta posvetovati z upravljavcem aplikacije OPZP. Vsi varnosti mehanizmi so narejeni tako, da omogočajo aplikaciji DOO delovanje v načinu brez povezave in preprečujejo morebitne zlorabe v tem načinu.

Na koncu opravilo uspešno prenesenim dnevnikom nastavi status "preneseni" in onemogoči urejanje teh dnevnikov v spletni aplikaciji DOO.

4.2.5 Administracija

Administracija se izvaja preko Swing aplikacije GemaAdministrator, ki je del informacijskega sistema GemaLogic in omogoča upravljanje s podatki sistema GemaLogic. Omogoča pregledovanje in urejanje vseh objektov v bazi, ki definirajo parametre sistema.



Slika 4.5: Konfiguracija dobavitelja podatkov

Za vzpostavitev in uporabo aplikacije so ključni naslednji pregledi:

- Uporabniki: Dodajanje novih uporabnikov, dodajanje pravic do različnih orodij, določanje, kateri pravi osebi pripada uporabnik.

- Merilna mesta: Pregled merilnih mest v sistemu, prikaz upravičencev do merilnih mest in njihove vloge na merilnemu mestu.

- Dobavitelji podatkov: Konfiguracija parametrov povezave s sistemom OPZP in definiranje digitalnih potrdil, ki se uporabljajo za varno prijavo v spletno storitev. (Slika 4.5)

- Sistemske nastavitve: Definirajo parametre sistema, kot so ime in verzija produkta, lastnik namestitve in razreševalnik digitalnih potrdil.

4.2.6 Povzetek

Izdelana aplikacija je zadostila vse naročnikove zahteve in projekt se je uspešno zaključil v predpisanem časovnem roku. Določene stvari bi se dalo boljše narediti, to se predvsem nanaša na izbiro tehnologij, na katerih se je aplikacija gradila. Zaradi naročnikovih zahtev in posledično tehničnih, časovnih in finančnih omejitev smo bili primorani izbrati tehnologije, na katerih bazira informacijski sistem GemaLogic.

V času izdelave aplikacije DOO je na voljo že bil aplikacijski strežnik JBoss 7, ki prinaša bistvene performančne prednosti v primerjavi z uporabljeno verzijo JBoss 4.2.2.GA in deluje z Javo 7, ki v primerjavi z Javo 6 prinaša dodatne pohitritve v delovanju. JBoss 7 omogoča enostavnejšo konfiguracijo varnostnih mehanizmov in omogoča več nastavitev na področju uporabe digitalnih potrdil, s katerimi ima JBoss 4.2.2.GA določene težave in omejitve.

Glavna težava aplikacijskega strežnika JBoss 4.2.2.GA je avtorizacija z uporabo digitalnega potrdila. Možna je zgolj avtentikacija, pri kateri prijavni modul prebere predefinirano vrednost iz digitalnega potrdila in jo upari z uporabniškim imenom. Za postopek avtorizacije dodatno potrebuje uporabniško ime, preko katerega poišče uporabnikove vloge v bazi podatkov. S stališča uporabnika je to odvečna informacija, saj se je že predstavil s svojim digitalnim potrdilom. Dodatna težava, ki pesti aplikacijski strežnik JBoss 4.2.2.GA, je uporaba zgolj ene varnostne domene znotraj enega modula.

Težava, na katero bi naleteli pri uporabi aplikacijskega strežnika JBoss 7, je ta, da za uparjanje digitalnega potrdila in uporabnika uporablja vrednost

DN (Distinguished name), medtem ko smo do sedaj uporabljali vrednost CN (Common name), v kateri je zapisano uporabniško ime. Ta vrednost ni nastavljiva, temveč je zapisana v kodi aplikacijskega strežnika, vendar imamo tudi za to pripravljen popravek aplikacijskega strežnika JBoss 7.

Glede spletnega vmesnika in ogrodja Seam smo uporabili skoraj najnovejšo verzijo. Za verzijo 2.2, katero smo uporabili pri aplikaciji DOO, so razvili še verzijo 2.3, katera je bila osredotočena na kompatibilnost z novimi tehnologijami in omogoča prenos aplikacij na novejše verzije aplikacijskega strežnika. Glede na to lahko rečemo, da so bile spletne tehnologije ustrezne, predvsem je tu potrebno poudariti, da Seam ponuja enostaven in robusten sistem integracije varnostnih mehanizmov.

Knjižnice gradnikov spletnega grafičnega vmesnika so bile deležne obširnih posodobitev, vendar ne igrajo nikakršne vloge pri zagotavljanju varnosti v aplikaciji. Vsekakor bi izbira novejših knjižnic pomenila lepši grafični vmesnik in odpravo določenih težav, ki smo jih imeli predvsem z Microsoftovim brskalnikom Internet Explorer, ki nam je do sedaj povzročal daleč največ težav. Tu lahko omenim, da je kompatibilnost brskalnika Internet Explorer v primerjavi s konkurenčnima brskalnikoma Firefox in Chrome bistveno slabša. Prav tako so performance občutno slabše, predvsem izvajanje JavaScript kode. Na žalost se moramo zavedati, da večina javnih podjetij in ustanov še vedno uporablja Internet Explorer kot privzeti brskalnik.

Dodatna varovalka, ki je nismo vgradili in bi še dodatno pripomogla k zagotavljanju varnosti, je ta, da bi se uporabniško ime prebralo iz digitalnega potrdila in bi bil uporabniku onemogočen ročni vnos uporabniškega imena. Na tak način bi lahko uporabnik dostopal do aplikacije samo s tistim uporabniškim imenom, ki je uparjeno z vrednostjo CN iz digitalnega potrdila, ki ga ima nameščenega v brskalniku.

Prehod na novejše tehnologije bi bila vsekakor dobrodošla nadgradnja sistema, vendar zaradi časovnih omejitev nemogoča. Takšna migracija tehnologij bi predstavljala tudi velik finančni zalogaj, zato ta korak ni bil ne predviden in ne financiran v sklopu projekta DOO. Tehnologije se hitro razvi-

jajo in napredujejo, tako bo tudi prehod informacijskega sistema GemaLogic na novejše tehnologije slej ko prej nujen.

Poglavje 5

Sklepne ugotovitve

V okviru diplomske naloge je bila razvita spletna aplikacija Dnevniki odjema odjemalca. Velik poudarek je bil na področju varnosti, saj bo aplikacija izpostavljena na medmrežju. Aplikacija bo vsebovala veliko zaupnih podatkov in razkritje le-teh bi imelo negativne poslovne posledice za naročnika.

Glavni prispevek k varnosti dajeta aplikacijski strežnik JBoss in ogrodje Seam. Slednje je tisto, ki nadgradi Java Authentication and Authorization Service (JAAS), predvsem ga pa poenostavi za uporabo. Z ogrodjem Seam je zagotavljanje varnosti bistveno enostavnejše kot zgolj z uporabo ogrodja JSF in tehnologije EJB. Seam upravlja s prijavo v aplikacijo, z implementacijo ene metode pa razvijalcu omogoča lastno implementacijo preverjanja uporabnika. Omogoča deklarativno varnost na nivoju strani in hkrati krpa luknje, ki jih pušča Servlet filter, z uporabo anotacij na Seamovih komponentah pa uvaja omejitve pri uporabnikovih akcijah in dostopu do podatkov. Hkrati omogoča vpeljavo konteksta v preverjanje uporabniških pravic, kar nam daje še več možnosti in boljši nadzor nad akcijami uporabnika. V določenih primerih ni dovolj vedeti samo, kateri vir uporabnik zahteva, ampak želimo vedeti, kaj uporabnik počne, in na podlagi tega se odločimo, ali odobrimo njegovo zahtevo ali ga zavrnemo.

Aplikacijski strežnik JBoss preko varnostne domene poskrbi za varen dostop do zrn EJB in spletnih storitev. Z anotacijami na razredih ali metodah

se določi, katere vloge zahteva od uporabnika, da se mu odobri dostop do želenega vira. S stališča razvijalca aplikacije je konfiguracija varnosti z uporabo anotacij JBossove varnostne domene zelo podobna temu, kar omogoča ogrodje Seam na spletu. Z uporabo knjižnice JBossWS postane razvoj spletne storitve izdelava zrna EJB s par dodatnimi anotacijami in tako tudi za varnost v spletnih storitvah skrbi vsebnik EJB.

Vpeljava digitalnih potrdil nam omogoča uporabo varne povezave preko protokola HTTPS ter vzajemno preverjanje pristnosti strežnika in odjemalca. Aplikacijski strežnik JBoss omogoča uporabo HTTPS Servlet vtičnika in samodejno preverjanje digitalnega potrdila, tako pri dostopu do spletnih strani kot pri dostopu do spletnih storitev.

Z integracijo vseh opisanih javanskih tehnologij smo pokrili glavne varnostne aspekte spletne aplikacije in realizirali naročnikovo zahtevo za izdelavo spletne aplikacije za varno elektronsko oddajajo dnevnikov o porabi zemeljskega plina na merilnih mestih. Uporaba digitalnih potrdil uporabnikom vliva dodatno zaupanje v varnost aplikacije in posledično zaupnost njihovih podatkov.

Literatura

[1] "Automation" (2013, Oktober) Dostopno na:
http://en.wikipedia.org/wiki/Automation

[2] J. D. Meier, Microsoft Corporation, "Improving Web Application Security: Threats and Contermeasures", *Microsoft*, 2003

[3] P. Kumar, "J2EE security for servlets, EJBs, and web services", *Prentice Hall*, 2004

[4] M. Cross, "Developer's Guide to Web Application Security", *Syngress Publishing, Inc.*, 2007

[5] "Application security" (2013, Oktober) Dostopno na:
http://en.wikipedia.org/wiki/Application_security

[6] "Java (programming language)" (2013, Oktober) Dostopno na:
http://en.wikipedia.org/wiki/Java_(programming_language)

[7] J. S. Fritzinger, M. Mueller, "Java Security", *Sun Microsystems, Inc.*, 1996

[8] S. Oaks, "Java Security", *O'Reilly*, 2. izdaja, 2001

[9] "Java security" (2013, Oktober) Dostopno na:
http://en.wikipedia.org/wiki/Java_security

[10] "The Java EE 5 Tutorial" (2013, Oktober) Dostopno na:
http://docs.oracle.com/javaee/5/tutorial/doc/bnbwj.html

[11] "Java Platform, Enterprise Edition (Java EE) Technical Documentation" (2013, Oktober) Dostopno na: http://docs.oracle.com/javaee/

[12] "Securing Applications" (2013, Oktober) Dostopno na: http://www.datadisk.co.uk/html_docs/java_app/jboss5/jboss5_securing_apps.htm

[13] J. Jamae, P. Johnson, "JBoss in Action", *Manning Publications Co.*, 2009

[14] "Configure Web Services" (2013, Oktober) Dostopno na: http://www.datadisk.co.uk/html_docs/java_app/jboss5/jboss5_config_web_services.htm

[15] D. Allen, "Seam in Action", *Manning Publications Co.*, 2008

[16] M. J. Yuan, J. Orshalick, T. Heute, "Seam Framework: Experience the Evolution of Java EE", *Prentice Hall*, 2. izdaja, 2009

[17] "Introduction to JBoss Seam" (2013, Oktober) Dostopno na: http://docs.jboss.org/seam/snapshot/en-US/html/Book-Preface.html

[18] K. D. Mann, "JavaServer Faces in Action", *Manning Publications Co.*, 2004

[19] M. Katz, I. Shaikovsky, "Practical RichFaces", *Apress*, 2. izdaja, 2011

[20] "RichFaces Developer Guide (PDF)" (2013, Oktober) Dostopno na: http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/pdf/richfaces_reference.pdf